REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEPTEMBER 2014	JOURNAL ARTICLE (Post Print)	APR 2013 – MAY 2014

4. TITLE AND SUBTITLE

PREDICTIVE FEATURE SELECTION FOR GENETIC POLICY SEARCH

5a. CONTRACT NUMBER
IN-HOUSE

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Steven Loscalzo, Robert Wright, Lei Yu

5d. PROJECT NUMBER
S2MA

5e. TASK NUMBER
MI

5f. WORK UNIT NUMBER
IH

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/Information Directorate
Rome Research Site/RISC
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/Information Directorate
Rome Research Site/RISC
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2014-044

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA Case Number: 88ABW-2014-2803
DATE CLEARED: June 6, 2014

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Automatic learning of control policies is becoming increasingly important to allow autonomous agents to operate alongside, or in place of, humans in dangerous and fast-paced situations. Reinforcement learning (RL), including genetic policy search algorithms, comprise a promising technology area capable of learning such control policies. Unfortunately, RL techniques can take prohibitively long to learn a sufficiently good control policy in environments described by many sensors (features). We argue that in many cases only a subset of available features are needed to learn the task at hand, since others may represent irrelevant or redundant information. In this work, we propose a predictive feature selection framework that analyzes data obtained during execution of a genetic policy search algorithm to identify relevant features on-line. This serves to constrain the policy search space and reduces the time needed to locate a sufficiently good policy by embedding feature selection into the process of learning a control policy. We explore this framework through an instantiation called predictive feature selection embedded in neuroevolution of augmenting topology (NEAT), or PFS-NEAT. In an empirical study, we demonstrate that PFS-NEAT is capable of enabling NEAT to successfully find good control policies in two benchmark environments, and show that it can outperform three competing feature selection algorithms, FS-NEAT, FD-NEAT, and SAFS-NEAT, in several variants of these environments.

15. SUBJECT TERMS
Genetic policy search; Feature selection; Dimensionality reduction; Reinforcement learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**STEVEN LOSCALZO**
U	U	U	UU	35	19b. TELEPHONE NUMBER (*Include area code)* N/A

Journal of Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

Predictive Feature Selection for Genetic Policy Search

Steven Loscalzo, Robert Wright, and Lei Yu

May 22, 2014

Abstract Automatic learning of control policies is becoming increasingly important to allow autonomous agents to operate alongside, or in place of, humans in dangerous and fast-paced situations. Reinforcement Learning (RL), including genetic policy search algorithms, comprise a promising technology area capable of learning such control policies. Unfortunately, RL techniques can take prohibitively long to learn a sufficiently good control policy in environments described by many sensors (features). We argue that in many cases only a subset of available features are needed to learn the task at hand, since others may represent irrelevant or redundant information. In this work, we propose a predictive feature selection framework that analyzes data obtained during execution of a genetic policy search algorithm to identify relevant features on-line. This serves to constrain the policy search space and reduces the time needed to locate a sufficiently good policy by embedding feature selection into the process of learning a control policy. We explore this framework through an instantiation called Predictive Feature Selection embedded in NEAT, or PFS-NEAT. In an empirical study, we demonstrate that PFS-NEAT is capable of enabling NEAT to successfully find good control policies in two benchmark environments, and show that it can outperform three competing feature selection algorithms, FS-NEAT, FD-NEAT, and SAFS-NEAT, in several variants of these environments.

Keywords Genetic Policy Search, Feature Selection, Dimensionality Reduction, Reinforcement Learning

Steven Loscalzo
AFRL Information Directorate, 26 Electronic Pkwy., Rome, New York 13441
E-mail: steven.loscalzo@us.af.mil

Robert Wright
Binghamton University, 4400 Vestal Pkwy. East, Binghamton, New York 13902
E-mail: rwright3@binghamton.edu

Lei Yu
Binghamton University, 4400 Vestal Pkwy. East, Binghamton, New York 13902
E-mail: lyu@cs.binghamton.edu

1 Introduction

Recent trends in autonomous systems research have illustrated the possibilities of deploying agents to complete dangerous or rapid-turnaround tasks, such as search-and-rescue, network intrusion defense, and data center management (Doroodgar and Nejat, 2010; Cannady, 2000; Servin and Kudenko, 2008; Tesauro et al, 2007). These systems require a control policy for each agent to follow that selects an appropriate response (action) for a given system state. Autonomous systems capable of learning their own control policies with limited manual intervention are becoming increasingly desirable as more complex tasks in dynamic and high-tempo environments are explored. Reinforcement Learning (RL) methods are of great interest because they require only the presence of a reward signal to indicate whether the chosen action was favorable or not, and do not need to know any system dynamics a priori.

While RL methods are attractive due to their highly autonomous nature, they all suffer from the so-called "curse of dimensionality": the computational time and interactions required to learn an optimal control policy is exponential in the number of sensors (state variables) that define an agent's view of the system (Bellman, 2003). This is expected to be an increasingly challenging problem as the environments where autonomous agents operate become more complex (i.e., contain more sensors of increased diversity). In this work we seek to address this problem as it applies to genetic policy search algorithms, one type of RL method. Genetic policy search algorithms are studied here since they have been shown to be broadly successful on a wide range of RL benchmark problems (Stanley and Miikkulainen, 2002; Böhm et al, 2004). In addition, they are capable of handling continuous and discrete state and action spaces without the need for additional abstraction devices as is typically the case for value iteration, policy iteration, and other direct policy search RL methods.

Feature selection has long been used by supervised learning for classification and regression problems. It can be used to automatically identify a compact set of relevant features that approximate the information content of the full data set (Guyon and Elisseeff, 2003; Liu and Yu, 2005). These techniques address the curse of dimensionality by removing features which appear irrelevant to the task at hand, and also excluding those redundant to other features that have already been selected. Irrelevant or redundant state variables may also be present in RL domains for various reasons. They may result from faulty sensors, availability of a richer suite of sensors than necessary to perform a single given task, or broadcast sensor readings from many agents co-located in the same environment. This observation gives rise to the analogue of feature selection in supervised learning: state variable selection in RL. For brevity, we will often use the term "feature" to refer to a state variable in the context of RL. This research area has received some attention in recent years (Hachiya and Sugiyama, 2010; Nouri and Littman, 2010), but feature selection has primarily been done with data collected off-line (i.e., before the agent begins operation in the system). Some works have incorporated feature selection with genetic policy search algorithms. The majority of these works solely consider evolved policy performance to guide the feature selection process, and do not incorporate information from observed sample data that could be used to improve feature selection decisions (Whiteson et al, 2005; Whiteson and Stone, 2006; Wright et al, 2011). Loscalzo et al (2012) does drive the selection process using observed sample data, but the feature relevance measure used is domain-specific and is unlikely to yield acceptable results in general.

We propose a generally applicable predictive feature selection framework for genetic policy search algorithms. This framework interleaves feature selection phases throughout the policy search process, using data observed during policy fitness evaluations to refine the selected feature subset. This mechanism restricts the policy search to consider policy

functions that contain only the most likely relevant state variable inputs. Constraining the search in this way allows a more rapid identification of good policies defined over a smaller set of state variables as compared to searching for the optimal policy in the entire input space. To this end, two challenging problems need to be overcome: when to alter the feature subset currently in use by the genetic policy search algorithm, and how to measure feature relevance. We offer an instantiation of the framework that addresses these challenges in the form of the Predictive Feature Selection embedded in NEAT (PFS-NEAT) algorithm. NeuroEvolution of Augmenting Topology (NEAT) was chosen as our base genetic policy search algorithm due to its widespread successes, and prior feature selection research that has been based off of it (Stanley and Miikkulainen, 2002).

We demonstrate the effectiveness of PFS-NEAT in two benchmark simulation environments. First, a racing domain described by continuous state and action spaces demanding that the agent make precise control decisions while knowing only what is visible from the driver's perspective, and second, a variant of the classical RL pole balancing domain, the double inverted pendulum balancing problem (Gomez and Miikkulainen, 1999), where the agent must learn a policy in a continuous state space using discrete actions to control the motions of a cart in order to balance two poles. We empirically show that PFS-NEAT is able to learn near optimal control policies in both environments even as the number of irrelevant features increases. The algorithm is also able to identify feature subsets that are comprised of higher fractions of relevant sensors as compared to FS-NEAT (Whiteson et al, 2005), FD-NEAT (Tan et al, 2009), and SAFS-NEAT (Loscalzo et al, 2012), which can cause PFS-NEAT to outperform these competitors. These experiments illustrate that PFS-NEAT is an effective feature selection algorithm that is suited to scaling up genetic policy search techniques to high-dimensional problems.

The rest of this paper is organized as follows: background on RL, feature selection, and relevant NEAT-based feature selection algorithms is presented Section 2. The PFS-NEAT algorithm is described in detail in Section 3. Section 4 provides an empirical study of the algorithm on two benchmark domains. Section 5 concludes this paper and outlines possible future directions.

2 Background

This work is positioned between the RL and feature selection domains, and so we briefly introduce both areas here. Section 2.1 introduces background on RL, and Section 2.2 introduces related work on scaling up RL by function approximation. Section 2.3 introduces background on feature selection and recent work on dimensionality reduction for RL. An introduction to the NEAT genetic policy search algorithm is given in Section 2.4. Section 2.5 introduces feature selection enabled variants of the NEAT algorithm, all of which are closely related to this work.

2.1 Reinforcement Learning

In general, Reinforcement Learning (RL) allows an agent to learn how to optimally complete a task in an environment from past experimence. The learning problem is typically modeled as a Markov Decision Process (MDP) and described by tuples $\langle \mathbf{S}, \mathbf{A}, P, R, \gamma \rangle$ (Puterman, 1994). In this work we make the general assumption that problems are described by

a factored state space $\mathbf{S}$ such that $\mathbf{S} = S_1 \times S_2 \times \ldots \times S_n$ for a problem with n state variables (Boutilier et al, 1999). Similarly, we consider the action space $\mathbf{A}$ to be represented as $\mathbf{A} = A_1 \times A_2 \times \ldots \times A_m$ for a problem with m distinct action variables. We denote the set of all state variables by $\mathbb{S} = \{S_1, S_2, \ldots, S_n\}$.

For every state $\mathbf{s} \in \mathbf{S}$ a point in the action space $\mathbf{a} \in \mathbf{A}$ must be selected by the agent, assigning a value to each of m simultaneous action decisions to be made in a state. Each state $\mathbf{s}$ is an n dimensional vector $\langle s_1, s_2, \ldots, s_n \rangle$ where each component s_i takes a value in the domain of its respective state variable (feature) S_i. The notation and semantics for each action $\mathbf{a}$ is analogous to this description. The transition function $P(\mathbf{s}'|\mathbf{s},\mathbf{a}) \in [0,1]$ governs the dynamics of the environment, or the probability of arriving in state $\mathbf{s}'$ after taking action $\mathbf{a}$ in state $\mathbf{s}$. The reward function $R(r|\mathbf{s},\mathbf{a},\mathbf{s}')$ is the expected value of the immediate reward $r \in \mathbb{R}$ given the transition from state $\mathbf{s}$ to $\mathbf{s}'$ on action $\mathbf{a}$. Both P and R are assumed to be unknown by the agent in our work.

The agent interacts with the environment in order to learn a control policy function, $\pi(\mathbf{s}) : \mathbf{S} \mapsto \mathbf{A}$, which describes the action to take in a given state. Each policy has an associated value function that defines the expected long-term value of taking action $\mathbf{a}$ in state $\mathbf{s}$ and following policy π thereafter: $Q_\pi(\mathbf{s},\mathbf{a}) = \mathbb{E}_\pi \left[\sum_{k=0} \gamma^k r_{t+k+1} | \mathbf{s}_t = \mathbf{s},\ \mathbf{a}_t = \mathbf{a}\right]$, where $\gamma \in (0,1]$ is the discount on rewards (Sutton and Barto, 1998). The agent seeks to learn the optimal policy function, π^*, which maximizes the expected value received when starting in any $\mathbf{s}$ and following π^* until a terminal state is reached. While learning π^*, the agent must repeatedly visit each state $\mathbf{s} \in \mathbf{S}$ to build an accurate estimate of the long term rewards received by taking each available action $\mathbf{a} \in \mathbf{A}$. This task becomes infeasible when either the size of the state space, $|\mathbf{S}|$, or action space, $|\mathbf{A}|$, is very large or infinite. This is a manifestation of the well known "curse of dimensionality" problem (Bellman, 2003), since the number of samples required to learn π^* is exponential in the number of dimensions of the state space.

2.2 Function Approximation

Function approximation techniques have allowed a measure of scalability to RL algorithms (Sutton and Barto, 1998), and have been receiving increased attention recently. Many of these approximation techniques for RL are centered on reducing the representation size of an accurate value function approximation. Function approximation methods use a model (such as a neural network or linear model) whose number of free parameters, k, is much smaller than the joint state-action space, $k \ll |\mathbf{S} \times \mathbf{A}|$, to approximate value functions. Several approximation approaches selectively grow a set of basis functions to account for error in the current approximation (Parr et al, 2007; Jung and Stone, 2009). Other approaches purposefully over-specify the number of basis functions in the set and use L_1 regularization methods to select a small number of these functions (Kolter and Ng, 2009; Petrik et al, 2010). Another basis function approach exploits the graph Laplacian on the state transition function of the given MDP to learn a minimal equivalent MDP to solve using RL (Mahadevan, 2005). While each of these methods has been shown effective in producing value function representations capable of approximating the optimal value function, they are all acutely sensitive to the number of state variables used to describe the state space. This is again due to the curse of dimensionality since the number and parameters of the basis functions depend on the size of the state space.

There are several other recent abstraction techniques that break an MDP into more readily solved sub-problems. Along the lines of hierarchical and structure learning (Dietterich, 1998), work has been done to find an abstraction to reduce the size of sub-problems to make

them easier to learn (Konidaris and Barto, 2009). This relies on the presence of a predefined abstraction library to select the most appropriate abstraction for a sub-problem, which may not be available in practice. Scalable RL techniques have also used learning from demonstration (Argall et al, 2009) to learn skills in a continuous state and action environment which can be put together to form an optimal policy (Konidaris et al, 2010). Selecting which skill to apply reduces the complexity of learning an optimal policy as compared to selecting an action at every state. Other methods in the approximate dynamic programming area (Powell, 2011) have made progress in solving models containing both discrete and continuous state variables (Kveton et al, 2006). These methods assume that the transition function is known, but this is not the case for our work, rendering these methods inapplicable. Overall, many different abstraction types exist for RL, but the size of $\mathbf{S}$ in either the original problem or a sub-problem can still prevent these methods from scaling to high-dimensional environments.

2.3 Feature Selection

Feature selection attempts to find a small subset of the original features defined by the problem to produce the best performance of a learning task. It is a widely used technique to battle the curse of dimensionality in supervised learning. In the context of RL, we consider each state variable to be a feature in this work. Many problems include irrelevant or redundant features in their descriptions, and incorporating these features can seriously degrade the performance of a learning agent. Irrelevant features are those which convey no information about the current learning task, and redundant features either partially or wholly duplicate information captured by other features. Models built from compact subsets of features which exclude irrelevant and redundant features typically generalize better than their full-dimensional counterparts because they reduce the effect of overfitting. Many algorithms exist for supervised feature selection (see comprehensive surveys by Guyon and Elisseeff (2003) and Liu and Yu (2005) for more details), and they can be categorized based on how their search and evaluation procedures operate.

In a problem with n features, there are 2^n different candidate feature subsets. An exhaustive search of this space to locate the smallest optimal feature subset would be too costly in any non-trivial case, so efficient search heuristics must be used. Many heuristics have been proposed for this task, such as beginning with an empty subset and growing it (sequential forward search), or starting with all the features and eliminating one or more at a time (sequential backwards search) (Devijver and Kittler, 1982). The search can terminate when either the subset reaches a desired size, or the candidate subset surpasses a specified quality threshold. At each step in the search, the quality of the candidate subset as well as the prospective next subsets must be evaluated to direct the search. The evaluation process can be broken down into three broad approaches: filter, wrapper, and embedded.

Filter algorithms evaluate the quality of feature subsets based on the intrinsic characteristics of the training data independent of any learning algorithm. This approach has been traditionally excluded from the RL scenario due to the lack of labeled training data, however, a few recent studies do suggest that filter techniques are applicable in RL (Hachiya and Sugiyama, 2010; Castelletti et al, 2011). These approaches use observations sampled from the environment to build up a data set, and then study the relationships between state, action, and reward information to estimate feature relevance. Wrappers use the performance of a learning algorithm on a feature subset to evaluate the subset. This approach is typically infeasible for RL because of the high computational and sample costs associated with learning as many policies as candidate subsets. Embedded approaches integrate model selection

with policy learning. These approaches enable the learning process to occur only once while still receiving the benefits of feature selection. The embedded approach is more promising than the wrapper approach for RL problems, since feature selection can be accomplished while learning a policy with little additional overhead.

Several methods have recently been proposed that explicitly reduce the dimensionality of the state space in an RL problem. Hachiya and Sugiyama (2010) proposed a mutual information approach to identify the features that the reward function is dependent on. This approach is reminiscent of filter feature selection methods since it requires a set of samples to be obtained from the environment a priori, which can be problematic. The sampling method must be able to gather samples from across the state space, which may not be possible for on-line learning environments if a good policy is not already available. The approach given by Castelletti et al (2011) bears some similarities in that it requires samples from a good policy to be available before feature relevance estimation can be conducted. Both of these approaches rely on a strong reward signal from the environment throughout the learning process, which may not occur in some RL problems with a sparse or delayed reward signal. Another related dimensionality reduction approach is given by Diuk et al (2009), but this method scales exponentially with the number of problem dimensions, preventing it from scaling to high-dimensional environments. Contrary to these approaches, our proposed algorithm, PFS-NEAT, applies an embedded approach designed to efficiently reduce the dimensionality of the state space in problems with high-dimensional discrete or continuous state spaces.

Several other RL algorithms also employ dimensionality reduction techniques, though their goal is not to find the smallest subset of features to learn the problem. Nouri and Littman (2010) propose such an approach where samples are gathered a priori and principal component analysis is performed to find low-dimensional subspaces to construct transition models which aid learning (Jolliffe, 2010). This approach does not guarantee that any feature will be excluded from the problem since a principal component is a linear combination of possibly all features. The number of transition model estimation points is exponential in the number of features and action values, preventing the method from scaling to high-dimensional problems. Other research along this direction attempts to learn a dynamic Bayesian network representation of the transition function, which may depend on a subset of features (Vigorito and Barto, 2009). While this work does not suffer from the exponential scaling problems of the approach proposed by Nouri and Littman (2010), it is reliant on the estimation of parameters for potentially many Gaussian models, which is challenging.

2.4 NEAT

The goal of this work is to enable higher policy performance in an RL domain through the use of feature selection to reduce of the effective size of the state space. As discussed in Section 2.3, a wrapper approach for RL is too costly, while a filter approach is not generally applicable. Therefore, we pursue an embedded approach in this work. The working mechanism of an embedded feature selection algorithm naturally depends on the choice of a learning algorithm. There are many reinforcement learning algorithms in existence, from simple Q-learning (Watkins and Dayan, 1992), to more contemporary learning algorithms such as Fitted Q Iteration (Ernst et al, 2005). We elected to use the direct policy search algorithm NeuroEvolution of Augmenting Topologies (NEAT) for its ease of use and ability to handle both discrete and continuous state and action spaces without additional abstraction layers (Stanley and Miikkulainen, 2002). NEAT is not the only candidate RL algorithm

which can learn in such diverse environments. Many recent function approximation algorithms have this ability, but introduce complicated steps in their execution to extend to such domains (Ernst et al, 2005; Nouri and Littman, 2010). Additionally, means for adapting existing learning strategies to domains with continuous action spaces have also been recently developed (Lazaric et al, 2007; Melo and Lopes, 2008; Pazis and Lagoudakis, 2009). We could have built our feature selection framework around one of these algorithms, but we preferred to keep the RL component as simple yet widely applicable as possible so we could focus on the challenges of feature selection.

Searching for an optimal policy for a problem with an infinite number of states to explore and infinite action values to select from (in the case of continuous state and action spaces) necessitates a function approximator to represent a policy. Neural networks (NNs) are efficient function approximators that can model complex functions to an arbitrary accuracy. In the past, using NNs in practice was limited by the difficult manual engineering that was required to construct a sufficiently usable NN. Neuroevolutionary approaches, which utilize genetic algorithms to automate the process of training and/or designing NNs, mitigate these drawbacks and allow NNs to be easily applied to RL domains (Sher, 2012). NeuroEvolution of Augmenting Topologies (NEAT) is an RL framework based on neuroevolution developed by Stanley and Miikkulainen (2002). The NEAT algorithm uses NNs to represent policy functions and conducts an evolutionary search of the policy function space to locate an approximation of π^*. By evolving both the network topology and the weights of connections between network nodes, Stanley and Miikkulainen (2002) showed that NEAT can solve typical RL benchmark problems several times faster than competing RL algorithms. Here we give a brief overview of the NEAT algorithm; more details can be found in the original work by Stanley and Miikkulainen (2002), and in the development of the NEAT-Q evolutionary function approximation algorithm by Whiteson and Stone (2006).

NEAT begins with a population of simple perceptron networks and gradually builds more complex ones through a process called *complexification*. Each network generated by NEAT describes a policy for an MDP. State features are provided to the networks as real-valued inputs, and the activation levels of output nodes prescribe the appropriate action values to use given the state. In every generation of the evolutionary process, the networks in the population are evaluated based on their ability to solve the MDP. In our setting, policy evaluation takes place by having an agent follow the policy in the environment, and uses an environment specific fitness function to assign a fitness score to the policy based on its performance. Since we are searching for π^*, which is the policy that maximizes the expected reward, the fitness function we use is the aggregate reward received by the evaluated policy. The most fit networks survive into the next generation, and derivative networks based upon these survivors are generated by the mutation operators. These mutation operators modify the weights of the edges and may also add topological elements such as new nodes and connections. These operators are activated with varying probability over the population of networks. The results of NEAT are NNs that are automatically generated, not overly complicated in terms of structure, and custom tuned for the problem at hand. The algorithm is guided by a number of parameters that are discussed in Section 4.1.3.

2.5 Feature Selection Methods Embedded in NEAT

This section discusses four prior extensions to the NEAT algorithm, all of which provide feature selection capabilities. They are distinguishable by their feature subset search strategies and subset evaluation measures.

2.5.1 FS-NEAT

One limiting issue with NEAT is that it assumes that all features provided by the environment are relevant and necessary, and it attempts to incorporate all of them into every NN in the population. Any irrelevant or redundant features will unnecessarily complicate the networks and slow the rate at which NEAT is able to derive an effective policy. Practically, evolution is slowed by considering mutations to unnecessary structures in NNs, which could lead to overly complex and inefficient networks. Worse, including such inputs increases the risk of overfitting to these features, which could produce very fragile policies that fail on previously unseen states. A first attempt at addressing this problem was proposed by Whiteson et al (2005) in the form of the Feature Selective-NEAT (FS-NEAT) algorithm. This algorithm embeds a feature selection process guided by evolution into the NEAT genetic search.

FS-NEAT differs from NEAT in that all NNs in the initial population start with a single connection between a randomly selected input and output. Features are randomly attached to the NNs in subsequent generations, governed by the add connection mutation operation. This allows the algorithm to evolve a potentially good feature subset during the regular NEAT policy search process. The evolutionary process treats the inclusion of another input node the same as adding any other connection to the network, such as from a hidden node to an output. One advantage of this algorithm is that it takes no additional computational time to evaluate features beyond the standard NEAT evolutionary search for fit networks. Relevant features are identified based on the fitness of the networks which include them, and high fitness scores cause those networks to propagate, thereby selecting the feature into the population. Allowing the subset search to be controlled by policy performance is an intuitive heuristic for conducting the subset search, but it also risks selecting unnecessary features into the set. If an irrelevant or redundant feature is present in a network which happens to perform well, that feature will go on to be included in subsequent generations, whether or not it contributed to the improved performance (which could have arisen due to random chance in noisy sensor readings, or a simultaneous mutation elsewhere in the NN). Such chance events become increasingly more likely as the ratio of relevant to irrelevant features decreases, causing NEAT to search through an artificially large policy space during its execution and slowing its arrival at a reasonably fit policy.

2.5.2 FD-NEAT

Feature Deselective-NEAT (FD-NEAT) follows an evolutionary feature selection strategy similar to FS-NEAT, but biases the subset search to favor larger selected feature subsets (Tan et al, 2009, 2012). The FD-NEAT algorithm begins with a fully connected network just as NEAT does, and then removes connections via a remove connection mutation operation during evolution. This feature subset search procedure is a recursive feature elimination search with a fitness-based evaluation criterion. Such a procedure is expected to perform well when there are few irrelevant features in the environment, and may outperform forward searches when there are strong feature interactions in the data (Guyon et al, 2002). Just as the FD-NEAT selection process is similar to FS-NEAT, it also shares similar benefits and limitations. The strategy is intuitive and adds no additional overhead to the NEAT policy search in order to perform the feature selection actions. Unfortunately, mutations may remove truly relevant features from the subset when the reduced subset happens to perform well during a fitness evaluation (for the same reasons as for FS-NEAT). This outcome may prevent the network from performing as well as possible. Additionally, evolved networks may contain many connections to an irrelevant feature. Prior policies are biased to work

with these extraneous features, and their removal can actually hurt the performance of the policy. This will generally lead to larger networks than necessary in the population, slowing the NEAT evolutionary search.

2.5.3 IFSE-NEAT

The Incremental Feature Selection Embedded in NEAT (IFSE-NEAT) algorithm was developed by the authors to address several of the limitations of the FS- and FD-NEAT algorithms (Wright et al, 2011). The main difference between these algorithms and IFSE-NEAT is the decoupling between the feature subset search and the policy search. A separate search should prevent extraneous features being selected by chance mutations, and would tend toward the most compact relevant feature subset being identified. This, in turn, would enable NEAT to avoid searching a larger space than necessary.

IFSE-NEAT performs an incremental forward search over the set of available features, adding a new feature every k generations of NEAT evolution. For a state space described by n features, the algorithm starts by performing n partial runs of NEAT in parallel, with each population using only a single unique feature. After k generations the resulting policies are compared and the feature used by the best performing partial run gets selected into the subset. Next, $n-1$ NEAT populations are created, each formed by connecting one of the unselected features to the best performing NN from the selected feature's population. Evolution continues on these populations in parallel for another k generations, and the best performing network indicates the current size-two subset of features to use moving forward. This process repeats until some number of generations elapses, a specific number of features is selected, or some other termination criterion is met. This method is successful because it allows for testing of each combination of features along the greedy incremental selection path for k generations before committing to a feature addition. Given enough computational resources, the algorithm can be run in similar time as the standard NEAT algorithm, but its sample requirements are $O(n^2)$ times greater than NEAT's sample requirements.

2.5.4 SAFS-NEAT

The Sample Aware Feature Selection embedded in NEAT (SAFS-NEAT) algorithm represents the most closely related work to the proposed PFS-NEAT approach (Loscalzo et al, 2012). SAFS-NEAT attempts to overcome the high sample complexity of the IFSE-NEAT algorithm by altering how features are evaluated. Instead of relying on policy fitness to indicate the next feature to select, sample observations are collected during NEAT evolution, and these samples are analyzed to yield the next feature to add to the current subset. SAFS-NEAT measures the correlation between state changes and the selected action to determine the set of controllable features (which are assumed to be relevant). This heuristic allows SAFS-NEAT to focus on searching policies which include features that clearly change as a result of actions, and works well in situations where actions have an immediate impact on the fitness of a policy. The SAFS-NEAT algorithm also enjoys the same sample complexity as NEAT, and only incurs a small time penalty as a result of measuring feature goodness. The algorithm enables NEAT to scale to problems with large state spaces without incurring the massive sample observation overhead that IFSE-NEAT does, but its feature evaluation measure is not general. There are many environments where irrelevant or redundant features are impacted by actions, and so would be highly selectable by SAFS-NEAT, and many other domains where relevant features cannot be controlled by an agent's actions, but figure highly into what the action decisions should be. For example, in a driving scenario, safe

turning speed significantly depends on both weather and vehicle weight features, but neither of these features can be controlled by the driver's actions.

The proposed PFS-NEAT algorithm includes a significantly more general feature relevance measure based on both relevance to reward and state transition information. This measure, combined with several other improvements, allows it to overcome the above limitations to become an efficient and general feature selection mechanism that compliments the performance of the NEAT genetic policy search algorithm.

3 Predictive Feature Selection - NEAT

In this section we detail the proposed Predictive Feature Selection embedded in NEAT (PFS-NEAT) algorithm. By using feature selection, we guide the policy search to consider only small sets of features, reducing the space of possible policies considered early in the search. The chief difficulty in performing feature selection during the policy search is finding the balance between including the appropriate features to be able to capture all information relevant to π^* while still constraining the search space so that an effective policy can be quickly identified. We divide the discussion of the algorithm along the lines of a typical feature selection algorithm, first detailing the subset search procedure in Section 3.1, followed by the feature evaluation measure in Section 3.2. We then provide an overview of the entire PFS-NEAT algorithm in Section 3.3, and conclude this section with a low-level exploration of parameters and implementation details in Section 3.4.

3.1 Subset Search

Deciding how to alter the feature subset is the primary challenge that must be addressed when designing a feature subset search strategy for use in a policy search algorithm. In supervised learning, the search task continues without delay until a termination criterion is met, since all of the training data is available to the feature selection algorithm. Data changes dynamically during a genetic policy search procedure because it is generated during the fitness calculation process when each policy in the population is evaluated. Since we expect that the policy improves over time, it is reasonable to expect that the data will also show new trends in feature relevance at different points during the policy search. This assumption necessitates that the feature subset search be embedded within the policy search. With this consideration in mind, we must now address the specific feature search strategy, and the timing of changing the feature space presented to NEAT.

As mentioned in Section 2.3, an exhaustive feature subset search would be too computationally expensive in any domain with many input features. In theory, any subset search mechanism can be used. However, we would like to reduce the impact of the feature search process on the policy search. NEAT follows a *complexification* paradigm, meaning that its population of NNs grows in complexity over time. The sequential forward search (SFS) procedure, widely used in feature selection for supervised learning tasks, intuitively matches this paradigm (Devijver and Kittler, 1982). Starting from an empty set, SFS iteratively expands the current subset by selecting the highest scoring feature(s) according to a given evaluation metric until a desired number of features are selected or some other termination criterion is met. SFS is particularly suitable for high-dimensional problems where a large portion of the features are irrelevant or redundant. SFS strategies cannot guarantee

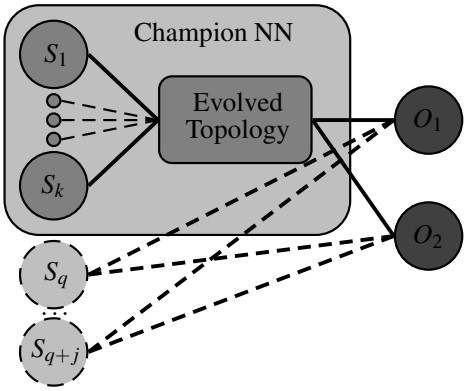


Fig. 1: This figure illustrates how newly selected features are incorporated into the current champion NN. Champion NN represents the best evolved policy function at some generation with features S_1 through S_k. The $j+1$ newly selected features, S_q through S_{q+j}, are introduced to the network through connections (dashed lines) to every output node.

the optimal solution to the feature subset selection problem (which would require evaluating 2^n subsets for a problem with n features), but this family of search strategies lead to computationally efficient heuristics on the order of $O(n^2)$. One limitation of simple SFS implementations is that they can be mislead when features only show relevance in combination with others (feature interaction), as in the XOR problem (Guyon and Elisseeff, 2003). To help alleviate this problem, we use the SFS variant Best-First subset search (Xu et al, 1988).

The Best-First algorithm searches all features currently not in the selected subset, and evaluates each one in turn with the selected subset using a given evaluation measure. The feature that causes the greatest subset score improvement is added to the subset, and the search repeats. The search terminates if no feature improves the subset score, which could result in zero, one, or several features being added to the currently selected subset. In the policy search context, after termination of Best-First, each of the newly selected features is connected to every NN in the population, as depicted in Figure 1. A newly added feature is directly connected to each of the m output nodes, resulting m new connections per added feature. The weights on each of the new connections are randomly selected, except for the champion (best performing) NN. The new connections in the champion network are assigned zero weight to preserve the previous best policy found in the population. We note that the search procedure may not find another relevant feature to add at a given step, and in this case, the NEAT evolutionary process continues with the previously selected subset.

The remaining consideration is the timing of the feature search. In theory, the collection of a single sample may allow a feature selection algorithm to discover an improved feature subset. In practice, running the evaluation procedure this frequently is likely to prevent the NEAT algorithm from having ample time to evolve competent policies that make use of the altered feature subsets, and would also be computationally expensive. Instead, we allow NEAT to evolve the currently selected feature subset until fitness *stagnation* occurs. In other words, until the fitness of the champion NN does not improve more than some user-defined threshold amount over some number of generations. This allows NEAT time to discover how a newly added feature may be integrated into an existing network to lead to better policies. Details on how to set the stagnation threshold parameters are deferred until Section 3.4.

We next address feature evaluation, the other important component of a feature selection algorithm.

3.2 Evaluation Measure

The success of any feature selection algorithm is dependent on its ability to correctly evaluate feature relevance. First, we must describe what feature relevance means in RL since there is no user specified class label or response variable, and relevance in supervised learning is typically described w.r.t. these notions. This discussion largely stems from the definition of feature relevance in RL provided by Castelletti et al (2011), and implicitly used by Hachiya and Sugiyama (2010). Feature selection in RL can be described as producing a mapping function $\rho : \mathbf{S} \mapsto \mathbf{S}^{\diamond}$, where the feature set $\mathbb{S}$ of $\mathbf{S}$ contains all state variables of some MDP M and the feature set of $\mathbf{S}^{\diamond}$ is $\mathbb{S}^{\diamond}$, with $\mathbb{S}^{\diamond} \subseteq \mathbb{S}$. More specifically, given some $\mathbf{s} \in \mathbf{S}, \rho(\mathbf{s})$ projects $\mathbf{s}$ to $\mathbf{s}^{\diamond} \in \mathbf{S}^{\diamond}$ such that any component $S_i \notin \mathbb{S}^{\diamond}$ is omitted from $\mathbf{s}^{\diamond}$, while any component $S_i \in \mathbb{S}^{\diamond}$ is exactly preserved. This function therefore causes an abstract MDP $M^{\diamond}$ to be created from M with state space $\mathbf{S}^{\diamond}$ (containing the Cartesian product of all state variables in $\mathbb{S}^{\diamond}$) with corresponding domain modifications to P and R as necessitated by this change. The optimal feature selection problem in RL requires searching for a minimal sized feature subset $\mathbb{S}^{\diamond}$ such that the mapping function $\rho : \mathbf{S} \mapsto \mathbf{S}^{\diamond}$ belongs to the model-irrelevant abstraction class defined by Li et al (2006). Given two distinct states $\mathbf{s}_1$ and $\mathbf{s}_2 \in \mathbf{S}$, and model-irrelevant abstraction ρ, $\rho(\mathbf{s}_1) = \rho(\mathbf{s}_2)$ implies $R(r|\mathbf{s}_1, \mathbf{a}, \mathbf{s}') = R(r|\mathbf{s}_2, \mathbf{a}, \mathbf{s}')$ and $P(\mathbf{s}'|\mathbf{s}_1, \mathbf{a}) = P(\mathbf{s}'|\mathbf{s}_2, \mathbf{a})$, or the agent has the same transition probabilities for all available actions $\mathbf{a}$. This guarantees that at least for Q-Learning (Watkins and Dayan, 1992), π^* found in $M^{\diamond}$ will also be π^* for the original MDP M (Li et al, 2006). Therefore, a relevant feature in this context is one necessary to compute R or P in $M^{\diamond}$. These guarantees only apply to Q-Learning, but the concept that relevant features are those which are informative with respect to the unknown R and P functions is applicable to all RL methods, policy search included.

Following this idea, we emphasize that there are features that are directly relevant to the unknown reward function R (*reward relevant features*), and those which are not directly relevant to the reward, but are relevant to reward relevant features (*transition relevant features*). The selection criterion is biased towards selecting reward relevant features because the reward is directly related to fitness, our objective measure. This means that an ideal feature selection algorithm would find the minimal subset of reward relevant features needed to describe R, but it does not need to identify transition relevant features for *every* feature's transition function. If we assume that the transition function $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ can be partitioned into n independent functions $P(\mathbf{s}'_1|\mathbf{s}, \mathbf{a}), P(\mathbf{s}'_2|\mathbf{s}, \mathbf{a}), \cdots, P(\mathbf{s}'_n|\mathbf{s}, \mathbf{a})$, then it is only necessary to learn the relevant features for each $P(\mathbf{s}'_k|\mathbf{s}, \mathbf{a})$ where S_k is a reward relevant feature, or a previously identified transition relevant feature. These descriptions will allow us to use the data collected during NN fitness evaluation for feature selection.

Each of the NNs in the population is evaluated in the environment in every generation of NEAT evolution. Fitness of each NN is given by aggregating the rewards received by following its policy. This behavior is necessary for the evolutionary process to function, since the composition of the next population is determined in part by the fitness scores of the networks in the current population. Over the course of each evaluation, a series of samples of the form $\langle \mathbf{s}, \mathbf{a}, r, \mathbf{s}' \rangle$ are observed, where $\mathbf{s}$ and $\mathbf{s}'$ are vectors of length n representing the current and next state feature values, respectively, r is the immediate reward, and $\mathbf{a}$ is a vector of length m containing each of m action decisions that were simultaneously made in

state $\mathbf{s}$. Note that the environment reports the value of each feature S_i whether or not the NN has S_i connected.

Given the data samples collected during fitness evaluation, we can structure them into several training data sets suitable for feature selection based on the discussion of feature relevance. First, consider the training data set D_r with features corresponding to all state and action variables in the environment, and the observed immediate reward of each sample as the class label. Performing feature selection on D_r will result in an estimate of reward relevant features given the currently collected samples. Next, let S_k be a reward relevant feature. Features relevant to the transition function of S_k can be identified by performing feature selection on a separate training data set D_k. We construct each D_k as we did with D_r, except we use $\mathbf{s}'_k$ for each sample instance as the class label instead of r. We repeat this process for all reward relevant features, and any other features found in each selection step to arrive at a subset of reward and transition relevant features to use in the policy search.

Given that we can determine class labels for a set of collected samples, we now have the ability to apply any supervised feature evaluation algorithm on the data to evaluate features. In this work, we used the symmetric uncertainty-based evaluation measure from the Correlation-based Feature Subset Selection algorithm, or CFS (Hall, 1999). Symmetric uncertainty is defined in terms of the entropy and conditional entropy of random variables,

$$H(Y) = -\sum_{y \in Y} p(y) \log_2(p(y)), \text{and} \tag{1}$$

$$H(Y|X) = -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log_2(p(y|x)), \tag{2}$$

where X and Y are random variables. Entropy is widely used in the information theoretic community to measure the uncertainty or unpredictability in a system. Symmetric uncertainty is then given by

$$SU(X,Y) = 2 \cdot \frac{H(Y) - H(Y|X)}{H(Y) + H(X)}, \tag{3}$$

which is the information gain measure ($H(Y) - H(Y|X)$), or the amount of information gained about Y after observing X, normalized by the entropy of X and Y. Information gain is biased towards features that have more values. Symmetric uncertainty corrects for this bias and so is more broadly applicable. Using the symmetric uncertainty measure, we can describe the scoring function used by CFS:

$$\text{score}(\mathbb{S}, \ell) = \frac{\sum_{S_i \in \mathbb{S}} SU(S_i, \ell)}{\sqrt{\sum_{S_i, S_j \in \mathbb{S}, i \neq j} SU(S_i, S_j)}}, \tag{4}$$

which assigns a numeric score to subset $\mathbb{S}$ with respect to class label ℓ based on the ratio of symmetric uncertainty between features in the subset and the class label, and all pairs of features in the subset themselves. This measure is capable of discriminating between relevant and irrelevant feature subsets, while also keeping redundant features out of the selected subset.

The overall feature selection methodology is given by the select-features function (Algorithm 1). This function takes as input the current set of selected features ($\mathbb{S}_{curr}$), which may be empty, and a set of samples D collected during the policy evaluation stage of NEAT. The specific sample collection strategy that we employ is discussed in Section 3.3. Line 3 of the algorithm creates a labelled data set D_r from D as described above. The CFS scoring

Algorithm 1 select_features($\mathbb{S}_{curr}, D$)

```
1: //S_curr: set of currently selected features
2: //D: data set of collected samples from environment
3: D_r ← filter(D, {s, a, r})
4: S_best ← CFS(S_curr, D_r)
5: if S_best = S_curr then
6:     //Count # selections of each transition relevant feature
7:     c[|S|] ← 0
8:     for S_k ∈ S_curr do
9:         D_k ← filter(D, {s, a, s'_k})
10:        S_k ← CFS(S_curr, D_k)
11:        for S_i ∈ S_k do
12:            c[i] ← c[i] + 1
13:        end for
14:    end for
15:    //Select the feature with the max count, breaking ties randomly
16:    S_best ← max_count(c)
17: end if
18: return S_best
```

algorithm is then run on D_r starting with the current feature subset and using the Best-First subset search heuristic to increase the selected subset size (Xu et al, 1988).

If the search does not identify any more *reward* relevant features to be added to the current subset, the select-features algorithm begins its search over the *transition* functions of each of the variables in the currently selected subset. For each previously selected feature S_k, we substitute each sample's next state $\mathbf{s}'_k$ as the class label and re-run the Best-First search with CFS scoring to discover the transition relevant features for S_k, denoted $\mathbb{S}_k$. We keep track of the frequency that each feature is a part of selected subset $\mathbb{S}_k, \forall S_k \in \mathbb{S}_{curr}$, and choose the maximum occurring feature to include in the returned subset. Ties are broken randomly between features that share the same maximum occurring frequency to ensure that at most one feature is selected from the transition relevant subset. This procedure differs from the reward relevant section, where a subset of size greater than one may be returned, to avoid adding many features at once to the population. Many different features may possess partial relevance to state transitions, and we have no a priori preference to select features corresponding to one feature's transition function over another's. We take each selection result as a vote and use the maximum consensus pick to limit the number of transition-relevant features that may be added in one call to the select-features algorithm. Additionally, a threshold value may be used to require higher scores before selecting features, which may be helpful in situations where noisy data causes inconsistent relevance measure results. In summary, the select-features algorithm will return a feature subset containing either:

1. previously unselected features that are found to be reward relevant, or
2. the previously unselected feature that was found to be most frequently relevant to the current set of transition functions, or
3. no features (an empty subset), indicating that CFS did not identify any remaining features as being relevant to D.

The returned subset $\mathbb{S}_{best}$ will be combined with $\mathbb{S}_{curr}$ to create $\mathbb{S}_{curr}$ for future generations. The selected feature subset used by the NEAT algorithm will therefore be monotonically non-decreasing in size after successive calls to the select-features algorithm.

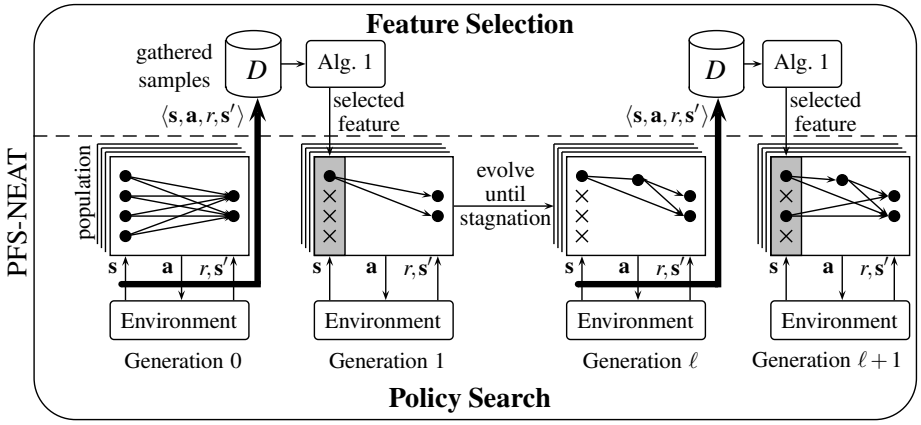


Fig. 2: Illustration of the Predictive Feature Selection embedded in NEAT (PFS-NEAT) algorithm. Samples collected from Generation 0 are used by Algorithm 1 to select the initial feature to be included. Sample collection and feature selection are then periodically carried out once the fitness of the population with the current set of selected features stagnates. This process repeats until adding an additional feature does not improve the fitness of the population.

3.3 PFS-NEAT Overview

Figure 2 provides a high-level overview of the entire PFS-NEAT process, focusing on the periodic calls to the feature selection component (discussed in the previous sections) from the policy search component. In this section, we will discuss this process in more detail, referring to Algorithm 2, a pseudo-code description of the PFS-NEAT process, throughout.

PFS-NEAT begins by initializing a NEAT population with fully connected networks (all features selected). NEAT-evolve is then called on this population, generating a set of samples D_i during the policy evaluation step of each NN. This data set is incorporated into the overall data store D, and the fitness of the best network in population Π is saved for use in determining stagnation. Once the stagnation criteria is met, then the feature selection block is entered. A subset of features $\mathbb{S}_{best}$ is returned from Algorithm 1 and this subset is added to the currently selected set of features. Any newly selected features are then combined with the best NN in Π (line 20) as in Figure 1 and the resulting policy π is used to re-initialize Π to connect just the features in $\mathbb{S}_{curr}$ to the networks. The policy search loop repeats until some termination criterion is met, such as a fixed number of iterations, or the fitness of the best NN exceeds some threshold.

Besides the previously discussed feature selection component, there are two technical considerations in a practical implementation of this algorithm: stagnation and sample selection. Stagnation effectively determines the maximum rate at which the feature subset, and hence the policy search space, can grow. In this implementation, stagnation is controlled by two user-defined parameters: a window size that covers some positive number of generations w, and the amount of change in the champion policy's fitness ε. The window size, w, forces the feature subset search to wait at least w steps in between calls to select-features. The population is only considered stagnant if the slope of previous w champion fitness values contained in the *champions* stack is less than ε. Setting w and ε is straight-forward to do in practice, and is discussed in Section 3.4.

Algorithm 2 PFS-NEAT(w, ε)

```
1: //w: stagnation window size
2: //ε: stagnation value threshold
3: D ← {}
4: 𝕊_curr ← {}
5: do-init ← true
6: champions ← empty-stack
7: //initialize the population
8: Π ← init-population(random fully-connected-network)
9: repeat
10:     D_i ← NEAT-evolve(Π)
11:     D ← incorporate-data(D_i)
12:     champions.push(bestNN(Π).fitness)
13:     if stagnant(champions, w, ε) or do-init = true then
14:         champions.clear()
15:         if do-init = true then
16:             do-init ← false
17:         end if
18:         𝕊_best ← select-features(𝕊_curr, D)   (Alg. 1)
19:         𝕊_curr ← 𝕊_curr ∪ 𝕊_best
20:         π ← combine(𝕊_best, bestNN(Π))
21:         Π ← init-population(π)
22:     end if
23: until /* termination criterion met */
24: return bestNN(Π)
```

Sample selection is conducted in Algorithm 2 by the incorporate-data function call (line 11), and requires further examination. Feature selection algorithms assume that the training data they are given are independent and identically distributed (IID), allowing the algorithms to select a compact subset of features that captures information from the data throughout the entire sample space. The IID assumptions do not hold in the data collected during policy evaluation, D_i, since each data sample $\langle \mathbf{s}, \mathbf{a}, \mathbf{s}', r \rangle$ leads to the next sample in the sequence and so are not independent, and they may not cover the entire sample space. This is especially valid in the early stages of policy search when policies may cause the agent to only visit states near the start state. Overcoming these challenges in general is out of the scope of this work, but we do propose a heuristic sample collection strategy, which we demonstrate in Section 4 works well in practice. During the evaluation of Π, all samples from the top five most fit networks are stored to improve diversity in stored samples. To assist in the identification of reward relevant features, we partition the overall sample store D into terminal and non-terminal samples, because the reward values for terminal states are often different, but less-frequently observed, than non-terminal states. Next, incorporate-data adds the samples from D_i to their appropriate bin in D. When D is used in select features (line 18), random sampling with replacement is used to balance the smaller partition (typically terminal states) with the larger partition so that the relevance measure is not biased towards the non-terminal states. If this balancing does not take place, the relative rarity of the different reward values seen from transitioning to terminal states in many domains causes features relevant to the reward variations to be overlooked, which hinders the policy search.

3.4 Parameter Selection

PFS-NEAT relies on the notion of policy fitness stagnation to determine when a population using a given set of features has likely reached its maximum potential. The stagnation criterion requires two parameters ε and w for this purpose. The complexity of the policy space should influence the selection of w and ε. For example, in a simple domain the search does

not require many generations to evolve a competent policy with a given feature subset and can therefore quickly progress from one subset to the next. In a domain obeying more complex transition and reward functions, more NN structure may be needed to be evolved before gains in policy fitness are observed for a given feature subset, and so the stagnation criterion should be set to delay changes to the selected subset. Determining the length of time to allow a population to evolve is a manifestation of the omnipresent exploration versus exploitation problem in RL (March, 1991). Allowing NEAT to explore more configurations using the current set of input features may allow it to discover a better policy, though there will be only marginal return on this investment of time after a certain point. At this point, exploiting the best known policy and adding a new feature to it may be a better direction for the policy search to follow.

Values of ε can be constrained in the range [0, 1] and then can be multiplied by the maximum fitness possible in an environment to arrive at a stagnation fitness threshold. If ε is zero, then stagnation only occurs when the difference in fitness over a span of w generations is zero. If set to one, then any fitness change, no matter how large, will be considered stagnant. Setting ε to something small is a reasonable compromise between the two extremes, and the algorithm tends to be insensitive to the exact setting. The window size w affects the quality of the stagnation estimate; the larger the w, the more accurate the stagnation check becomes. In practice, most window sizes produce sufficiently accurate estimates, so this parameter is also easy to set. The two parameters do indirectly control how frequently features get added to the subset, since a new feature can only be included after the previous population reaches stagnation. If a very large window is set, then many generations must elapse before a stagnation check can succeed, delaying feature selection. To allow PFS-NEAT to include features at a reasonable rate, small w and nonzero ε are preferred.

The stagnation threshold has additional implications on the NEAT policy search. If w and ε are set such that many generations must elapse before the population is considered stagnant, the champion NN may evolve additional structure that does not change the fitness, but is included in subsequent generations. When the subset does change, the NEAT search will be needlessly complicated by this additional structure, slowing future progress. Overcomplication can also happen prior to stagnation; NEAT may improve the fitness of its population using towards a local minimum in the fitness landscape for the current feature subset. This local minimum may be "deceptive," meaning that it does not clearly lead to the global optimal policy (Goldberg and Richardson, 1987), and the feature subset should have been changed earlier to escape the local optimum. This is a challenging problem that is beyond the scope of this work, and has been addressed in recent studies that encourage diversity in the search space to avoid becoming trapped in these local minima (Mouret and Doncieux, 2012). Coupling the feature selection process to other mechanisms that address deceptive fitness landscapes such as multi-objectivism (Knowles et al, 2001), coevolution (Cliff and Miller, 1995), or evolutionary searches that optimize on novelty instead of fitness (Lehman and Stanley, 2011) may alleviate this issue and are an open area of future study.

4 Empirical Study

In this section we provide an empirical study that measures and demonstrates PFS-NEAT's ability to learn in over-specified problem domains. This study is conducted in two benchmark environments described by both designed relevant and additional features. We show that PFS-NEAT can learn good policies by selecting small subsets of relevant features in both of these environments. Section 4.1 describes the problem setup of both the robot auto

Name	Informative	N(0.5, 0.25)	Lagged	Broadcast	Total
$D_{(10,0)}$	10	0	0	0	10
$D_{(10,30)}$	10	10	10	10	40
$D_{(10,60)}$	10	10	10	40	70
$D_{(10,90)}$	10	10	10	70	100

Table 1: RARS scenarios under study, decomposed into number of known informative sensors, and the three categories of known irrelevant or redundant features.

racing simulator and the double inverted pendulum balance environments. It also includes parameter settings for all algorithms included in the study. Section 4.2 presents the results of our study and is broken into three parts. The first two correspond to the two problem environments mentioned above, and the third provides a summary of our main results.

4.1 Experimental Setup

4.1.1 Robot Auto Racing Simulator

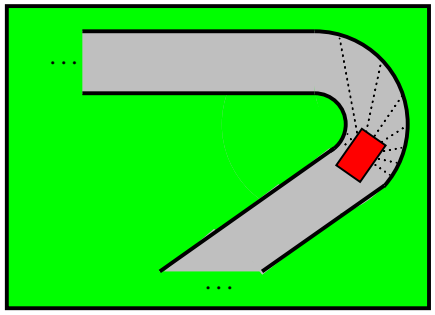


Fig. 3: The range finder sensors measuring the car's position relative to the nearest track walls.

Fig. 4: Overhead view of the track used in this study, included in the RARS distribution.

Our first experimental domain is a Java port of the Robot Auto Racing Simulator[1] (RARS) racing simulation environment. The goal of this problem is to drive a car around a track as quickly as possible, while keeping the car on the track. Each state of the environment is defined by a set of 9 position sensors and the car's speed. The sensors evenly span the 180° area in front of the car as depicted in Figure 3, and measure the distance from the car to the nearest track wall along that direction. The set of 10 position sensors and velocity are considered relevant, or informative, features. In each state, the learner must use these features to determine correct values for two continuous actions, the next desired vehicle speed and direction.

All RARS experiments were conducted on the track shown in Figure 4, which contains a representative sampling of driving situations including straightaways and turns in either direction that are both sharp and gentle. We defined the reward function of the RARS environment to be the distance traveled in a time step, or -5 if the car leaves the track area. We limit each policy evaluation to 3,000 time steps in the environment which is sufficient for a

[1] Project located at: http://rars.sourceforge.net

fit policy to complete approximately 1.5 laps of the track. The fitness function therefore reports the total distance traveled by the car in feet, -5 if the car leaves the track before the end of the evaluation. The penalty of -5 is minor compared to fitness scores of high-performance policies ($\sim$30,000), but it is instructive early on in learning when the car does not travel far before crashing.

Beyond the above set of sensors that are designed to be relevant to the driving task, we also introduce several types of additional features to better capture real world sensing challenges. Gaussian random variables with mean 0.5 and standard deviation 0.25 are used to simulate irrelevant sensors readings (e.g., engine coolant level, radio frequency, air temperature, etc.). Lagged sensor readings are also provided and report the informative sensor readings from five time steps earlier, notionally caused by sensor processing delays or some minor system fault. During the first five steps of the simulation, these sensors report a zero value. The lagged sensors may provide some relevance to the policy search algorithm. For instance, the policy may be able to improve itself by being able to measure velocity changes over this period. Despite this fact, we consider them to be part of the additional feature set in our evaluation since the driving task can be learned from the original set of designed relevant features. Finally, readings from other vehicles in the environment are provided to the agent to simulate broadcast interference in a manner similar to the experimental setup of Nouri and Littman (2010). Each car reports the velocity and nine distance readings in the same fashion as the controlled car. These other cars all follow the top performing policy found by NEAT in the environment when only the relevant set of sensors was used. This policy's performance is commensurate with the best seen manually developed policies for this environment. All cars (real and virtual) begin equally spaced around the track to provide diversity of sensor inputs. For simplicity, the cars do not physically interfere with each other on the track. All sensor readings, or feature values, are normalized (or clipped for Gaussians) to report values in [0, 1].

The Gaussian random, lagged, and other vehicle sensors are collectively referred to as the *additional* sensors in our study and note that some of the sensors may improve learning performance on their own or when considered in combination with the designed informative sensors. We label the domains under study by $D_{(\#\ informative, \#\ additional)}$. The four domain variants we consider in the RARS environment are listed in Table 1, where we achieve scaling by adding more cars on the track and reporting their broadcast sensors as described above. The decision to scale the number of broadcast sensors was made because it represents the most realistic possibility of the sensors we used, and provides a challenge to our algorithm. Increasing the number of lagged sensors (by altering the delay time) increases redundancy among features in the feature set, allowing the PFS-NEAT algorithm to easily detect and avoid adding many of them to the set. Loscalzo et al (2012) showed that scaling the number of Gaussian random sensors provided very little impact on SAFS-NEAT, and our preliminary study revealed that more Gaussians do not present a challenge to PFS-NEAT either. As seen in the experimental section of that work, random sensors do have a strong negative impact on FS-NEAT. Including an irrelevant feature may not harm the policy in one generation, but subsequent improvements will need to work around unpredictable values from that sensor, causing the algorithm to become stuck in local optima. When there are many of them present, this situation becomes more likely, reducing the probability of a successful FS-NEAT performance. Broadcast sensors sometimes report values that look quite relevant to the controlled car's performance, and are therefore more challenging for our method to classify as informative or not. On the other hand, the other algorithms in our study are better suited to evolve around these features than Gaussian random ones because

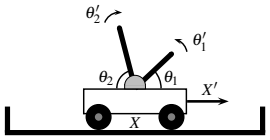


Fig. 5: Double inverted pendulum balancing problem with all six relevant features labeled.

Table 2: Double pole balance scenarios under study, decomposed into number of known informative sensors, and the three categories of known irrelevant or redundant features.

Name	Informative	N	Lagged	Broadcast	Total
$D_{(6,0)}$	6	0	0	0	6
$D_{(6,18)}$	6	6	6	6	24
$D_{(6,36)}$	6	6	6	24	42
$D_{(6,54)}$	6	6	6	42	60

evolved NNs can more easily compensate for the regular patterns of the broadcast sensors than the Gaussian random values.

4.1.2 Double Inverted Pendulum Balancing

The Double inverted Pendulum Balancing problem (DPB) is a standard RL benchmark problem (Gomez and Miikkulainen, 1999). The agent is required to balance two poles with different lengths and masses that are attached at one end to a movable cart, as illustrated in Figure 5. Control is achieved by selecting one of three discrete actions in each time step: move left, move right, or do not move. In addition to balancing the two poles, the agent is required to keep the cart within a small length of track. The problem is considered solved if the agent can learn a policy which keeps the poles upright for 100,000 time steps while keeping the cart within the restricted track. The reward function returns 1 while the poles are balanced, and 0 if either pole falls over or the cart strays beyond the boundaries of the track segment. Once either pole falls, the policy evaluation is terminated. The fitness function is simply the number of time steps that the poles remained balanced and the cart did not stray beyond the designated area, and so is limited to 100,000 in our experiments.

Each environmental state is described by six continuous inputs: position of the cart (X), velocity of the cart (X'), the angle between each pole and the cart (θ_1 and θ_2), and the angular velocities of the poles (θ_1' and θ_2'). We again simulate a more sensor rich environment by adding Gaussian random sensors, lagged sensors, and broadcast values from other cart-pole systems to the state space, as done in the RARS environment in Section 4.1.1. Broadcast sensors are reported by other (virtual) cart-pole systems in the environment, each contributing six feature readings from the sensors described above. These virtual cart-pole systems follow a policy learned by NEAT which was able to balance the poles for at least 100,000 time steps. Each virtual cart-pole system begins from a slightly different starting location, and the poles do not fall during execution. We experimented with increasing numbers of additional features as shown in Table 2, scaling the number of virtual cart-pole systems to achieve greater feature sizes. The rationale behind scaling these sensors as opposed to the random or lagged sensors is the same as for the RARS environment.

4.1.3 Algorithm Parameter Settings

This study compares results for five algorithms: NEAT, FS-NEAT, FD-NEAT, SAFS-NEAT, and PFS-NEAT. NEAT serves as the baseline learner which does not perform any feature selection. FS-NEAT, and FD-NEAT are existing feature selection algorithms that use an evolutionary search heuristic to select features, as described in Sections 2.5.1& 2.5.2 respectively. SAFS-NEAT controls the selected subset externally from the evolutionary process,

as described in Section 2.5.4. PFS-NEAT is our instantiation of the predictive feature selection framework, and is described in Section 3.3. Though IFSE-NEAT was discussed in Section 2.5.3 and can be expected to perform well in these domains based on past empirical evidence Wright et al (2011), it is not included in this study because it runs $O(n^2)$ additional runs of the NEAT algorithm during the course of execution to determine feature relevance. These extra evolutionary steps put the algorithm on unequal footing with the others, and so we do not include it in this study.

All five algorithms are based on the NEAT genetic policy search algorithm, and share a number of parameter settings that control the evolutionary search. The population size p must be large enough to promote genetic variation amongst different candidate NNs, though keeping it small will reduce the time required to evaluate the population in each NEAT-EVOLVE function call. Setting it to 100 provided a fair balance between allowing NEAT to explore new NNs, while perpetuating previously evolved NNs. Experimental results revealed only minor differences in performance with either smaller or larger population sizes. The top 20% of the population was propagated unchanged into the next population. NEAT population members are eligible to reproduce if their compatibility scores are greater than 0.5. Weight parameters control how excess, disjoint, and matching genes contribute to the compatibility score in NEAT (Stanley and Miikkulainen, 2002). We set the excess coefficient to 1.0, disjoint to 1.0, and matching to 0.04 in all experiments, and found that the algorithms are largely insensitive to variations in these values for the experimental domains considered. Activation functions on the input neurons are linear, while all other activation functions are sigmoid. Recurrent NNs are disallowed in all experiments because effective policies were found without the need for recurrent networks, and allowing them would slow the policy evaluation process. Since evolution is a stochastic process, all experiments were run 100 times, each with a different random seed, and the results shown in the figures of Section 4.2 are the averages across these 100 runs.

There are additional parameters which are environment specific. For RARS, it was found that linear networks could produce a successful policy, so the add neuron mutation was turned off. To randomly add features into the networks, FS-NEAT used an add connection mutation rate of 0.01, a value found to work best across all RARS scenarios via a parameter sweep in $\{0.01,0.02,0.05,0.10,0.20\}$. This value means that approximately one percent of all possible connections which can be added for each network will be added during the mutation step. Larger add connection rates could lead to reaching maximum performance more rapidly, but this maximum was lower than the one attained using 0.01. Evolution was permitted for 300 generations to give ample time for convergence. FD-NEAT requires the use of a remove connection weight which was set to 0.01 in RARS, the best performing value found in $\{0.001, 0.005, 0.01, 0.02, 0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90\}$. SAFS-NEAT and PFS-NEAT used a window size of $w = 10$ generations, and set $\varepsilon = 0.001$ to control stagnation, where the possible fitness values of an NN in the RARS test environment are in the range of [0, 30,600]. For DPB, all five algorithms set the add connection mutation rate to 0.02 and add neuron mutation to 0.01. Again, this value was found for FS-NEAT via parameter sweep in the same range as for RARS. FD-NEAT set its remove connection rate to 0.70, and was chosen via parameter sweep from the same set of values as for RARS. Linear networks can also be successful in the pendulum balancing environment, but a non-zero add neuron mutation rate was found to be useful to all algorithms. A general policy was learned by using a problem set of 100 instantiations of the cart-pole system. Each time, the cart was positioned in the center of the track, while the poles were given different initial angles and velocities. The problem instances are all solvable and identical across runs. SAFS-NEAT and PFS-NEAT again used a window size of $w = 10$ generations, and set $\varepsilon = 0.001$. The possible

fitness values of an NN are in the range of [0, 100,000] for this domain. SAFS-NEAT and PFS-NEAT employed an expansion factor of 1.2 to increase the window size after each selection round (i.e., the first stagnation check occurs after 10 generations, the second must wait for at least 12 generations, etc.). This encourages adequate sample collection over the course of evolution.

All algorithms in this study are based on the ANJI[2] implementation of NEAT, and Weka's implementation of CFS[3]. We have provided implementations and documentation for all algorithms in this study at `https://github.com/sloscal1/PFS-NEAT`. It should be noted that the mutation operator rates have different meanings in the ANJI code than in the standard NEAT description. Mutation rates were defined as the fraction of the population that would experience the mutation in the original NEAT description, but ANJI interprets the rates as the probability that any single possible mutation in a network will happen. This produces another variation from canonical NEAT to ANJI, multiple topological mutations are allowed on an NN in one evolutionary generation in ANJI, but only a single topological mutation was permitted in the original NEAT algorithm.

4.2 Results and Discussion

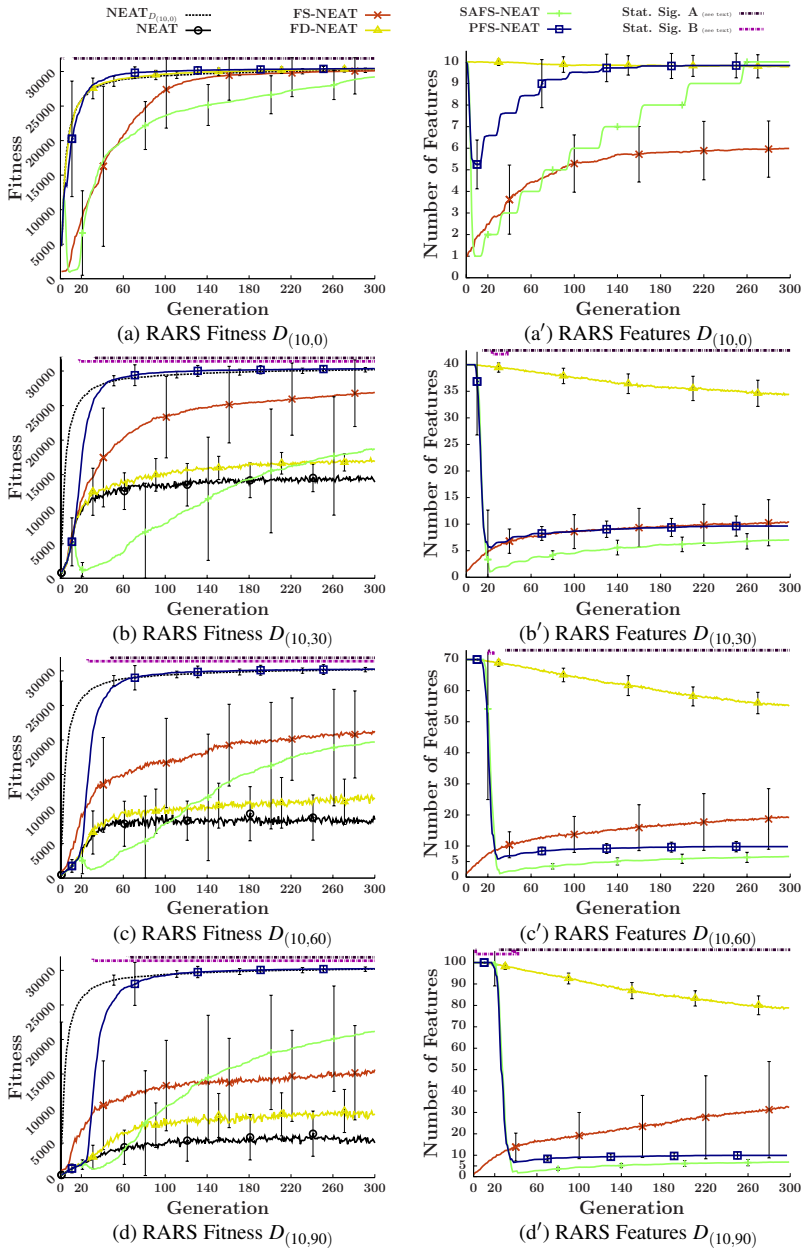
4.2.1 Robot Auto Racing Simulator

The left column of Figure 6 contains performance results of the three algorithms in the RARS environments as the number of additional sensors increase from 0 in panel (a) to 90 in panel (d). Scenarios with additional sensors ((b) – (d)) also show the performance of NEAT on $D_{(10,0)}$ for reference. To better gauge the significance of theses results, the plots include two statistical significance tests. The "**Statistical Significance A**" curve (two tick marks) is visible at the top of each fitness plot for those generations when PFS-NEAT is either statistically significantly better or not significantly distinguishable from NEAT on $D_{(10,0)}$ with $p < 0.01$. The "**Statistical Significance B**" curve (one tick mark) is visible near the top of each fitness plot for those generations when PFS-NEAT is statistically significantly better than the other three feature selection algorithms in the given scenario with $p < 0.01$. All significance tests (in both experimental domains) were done using Welch's t-test with unpaired samples and unequal sample variance.

The connection between fitness value and the policy quality is abstract in these results, but subjective observations can be made regarding the effectiveness of policies with different fitness values. Policies with fitness at around 1,000 or less typically correspond to cars that drive directly into walls with little or no steering capability. Once policies achieve scores around 20,000, they can usually drive a complete circuit on this track, though do so at a near constant velocity. Fitness values around 30,000 indicate that a policy has learned to quickly navigate turns of varying sharpness and accelerate through straightaways. The maximum fitness values obtained by the learning algorithms are similar to what manually programmed drivers can achieve on these tracks, and these drivers consistently appear to make smart driving choices.

[2] http://anji.sourceforge.net

[3] http://www.cs.waikato.ac.nz/ml/weka



(a) RARS Fitness $D_{(10,0)}$ (a′) RARS Features $D_{(10,0)}$

(b) RARS Fitness $D_{(10,30)}$ (b′) RARS Features $D_{(10,30)}$

(c) RARS Fitness $D_{(10,60)}$ (c′) RARS Features $D_{(10,60)}$

(d) RARS Fitness $D_{(10,90)}$ (d′) RARS Features $D_{(10,90)}$

Fig. 6: Plots (a) – (d) (left column) denote the average fitness of the algorithms in comparison during the first 300 generations of evolution for each of the RARS scenarios described in Table 1. Plots (a′) – (d′) (right column) show the corresponding selected feature subset sizes of the four feature selection algorithms in comparison. Curves in all plots follow the key at the top of the page. Plots (b) – (d) contain a dotted reference curve replicating the performance of NEAT on $D_{(10,0)}$ (the scenario using only informative features). The Stat. Sig. A and B curves are shown when their conditions are met with $p < 0.01$ in a given generation (see text for details). Error bars denote the standard deviation of the 100 runs at each point.

The negative impact of the additional sensors on the NEAT policy search is clearly illustrated by the increasingly large gap in performance between the reference curve and NEAT's performance curve as the scenarios introduce more additional sensors. In contrast, PFS-NEAT is able to evolve policies that are primarily composed of informative sensors, allowing the policy search to consistently locate policies that are statistically indistinguishable from the reference curve by generation 70. All feature selection algorithms perform well in the case when no additional sensors are considered (Figure 6(a)), but when additional sensors are included, the benefits of FS-NEAT, FD-NEAT, and SAFS-NEAT are diminished. While these other feature selection algorithms outperform NEAT at generation 300, PFS-NEAT statistically significantly outperforms all competitors after generation 40 in scenarios (b) – (d). FS-NEAT reaches an intermediate performance between the other two algorithms, scaling better than NEAT but worse than PFS-NEAT. The standard deviation bars on the plots also deserve some explanation. For FS-NEAT and SAFS-NEAT these bars remain relatively large throughout learning. Both algorithms heavily depend on stochastic influences to select features: FS-NEAT directly via evolutionary performance, while SAFS-NEAT requires varied samples observed during evolution to guide the subset search. In some runs, good policies are evolved early (and for SAFS-NEAT diverse and informative samples are collected) and lead to high performance policies. In other runs, additional features get selected early by chance, and prevent the policy search from evolving policies with even the fitness of the baseline NEAT algorithm. These divergent outcomes lead to large standard deviations. FD-NEAT is very similar to NEAT and makes smaller changes to the subsets in this environment than the other selection algorithms. This allows the performance to be more consistent than FS-NEAT or SAFS-NEAT, but typically only slightly better than the NEAT algorithm. PFS-NEAT starts with large deviations, as different relevant features are selected early in learning, but these deviations shrink over time. This is due to the different runs converging on a similarly good set of relevant features, which in turn produces consistently fit networks. These results demonstrate the need for feature selection in over-specified RL domains.

Let us next consider the performance of baseline NEAT algorithm as the number of sensors in the environment grows. It performs significantly worse as more additional features are added to the problem because the evolutionary search has a lower probability of making several good mutations simultaneously to improve the NNs. Only one or even a few prosperous mutations may be insufficient to drive the fitness of an NN high enough to guarantee its inclusion in the subsequent generation. Failure to survive to the next generation will cause these good mutations to die out from the population, preventing NEAT from progressing towards an optimal policy. FS-NEAT and FD-NEAT are able to perform better than NEAT when considering additional sensors by virtue of incorporating fewer of them into their networks, requiring fewer simultaneous successful mutations to cause fitness improvement. They also clearly shows a sensitivity to the number of additional features included in the problem, and attain increasingly lower fitness values as more of them are present in the problem. For FS-NEAT, this is due to the high probability of a mutation connecting an additional feature into the network, forcing later generations to evolve around these errant features. FD-NEAT must remove increasing numbers of features to increase chances of successful policy searches, and it runs the risk of removing informative features in the process. SAFS-NEAT causes a small number of sensors to be selected in this environment, but it does not always succeed at selecting informative sensors which we discuss below. PFS-NEAT's feature selection strategy is not dependent on the number of additional features since they are given low evaluation scores. Therefore, they are not selected into the feature subset, regardless of their prevalence in the environment.

The right column of Figure 6 shows the progression of the selected feature subsets during learning in each of the four scenarios (a′) – (d′). These plots serve to show the size of the selected feature subsets and when the feature subset is changed during the evolutionary process. Note that the scales of the y-axes differ according to the total feature set size in each scenario. In these plots, the "**Statistical Significance A**" curve is visible in those generations when the fraction of informative sensors in PFS-NEAT's selected subset is significantly greater than the fraction of informative sensors in each of the other three algorithm's subsets (at $p < 0.01$). "**Statistical Significance B**" is visible when the fraction of informative and lagged sensors in PFS-NEAT's selected subset is significantly greater than any of the other algorithm's subsets (at $p < 0.01$). These measures convey the quality of the selected subsets, accounting for the possibility that the lagged sensors may in fact be useful to the policy search process. These curves do not appear in (a′) since all features are informative in that scenario. In plots (b′) – (d′) we see that PFS-NEAT consistently selects subsets containing more informative sensors than any other algorithm for much of the evolutionary process, though other algorithms include enough lagged sensors in their subsets that their compositions in terms of informative and lagged sensors are not statistically significantly worse than PFS-NEAT. The quality and size of the selected subsets underpin why PFS-NEAT is able to outperform the other algorithms in the scenarios involving many additional features. The subset curves in (b′) – (d′) also explain why PFS-NEAT lags behind the reference curve early in the search in fitness plots (b) – (d). We see that PFS-NEAT delays its initial subset selection until a threshold amount of samples has been collected to lower the chances of finding a spurious pattern during the relevance measurement step. The algorithm then selects features as quickly as permitted by the stagnation parameters until nearly all features are selected. In scenarios $D_{(10,0)}$ and $D_{(10,60)}$ we see that the window parameter dominates the selection process of both PFS-NEAT and SAFS-NEAT. If the window was larger, the subsets would grow more slowly along with overall fitness. If the window is shrunk too much below its current value of 10, the algorithm increases its chances of selecting irrelevant features due to poor sample diversity. SAFS-NEAT is forced by design to select one feature at a time, and so more slowly reaches a quality subset, as can be seen most clearly in (a′). FD-NEAT removes features from its selected subset over time, allowing the policy search to have more success than the baseline NEAT algorithm, and may continue to improve if more generations are permitted. FS-NEAT slowly builds the selected subset throughout the policy search, resulting in a larger subset than PFS-NEAT and SAFS-NEAT in settings $D_{(10,60)}$ and $D_{(10,90)}$.

Though PFS-NEAT and FS-NEAT arrive at a similarly sized subset in the $D_{(10,30)}$ scenario, the composition of the subset leads to the difference in fitness observed in Figure 6(b). We examine these differences in more detail in Figure 7. We first note that FD-NEAT is on a larger y-axis scale than the other algorithms since it selects a larger subset size in this scenario, and that baseline NEAT is not shown since it does not change the subset size during learning. We next point out that PFS-NEAT and FS-NEAT select the highest and second highest numbers of informative sensors in the environment, respectively, and have the best performing results in Figure 6(b). PFS-NEAT also keeps out Gaussian and broadcast sensors, giving a performance edge to that algorithm. We see that SAFS-NEAT has a tendency to select lagged sensors at about the same rate as informative ones in this scenario, causing the policy search to have difficulty evolving a reasonably fit network in this environment. FD-NEAT removes all types of sensors at roughly equivalent rates. The composition of FD-NEAT's selected subsets in $D_{(10,60)}$ and $D_{(10,90)}$ (not shown) reveal a slight preference for retaining more informative features than the other types. This outcome supports the above hypothesis that with more generations, FD-NEAT may also arrive at a fit policy.

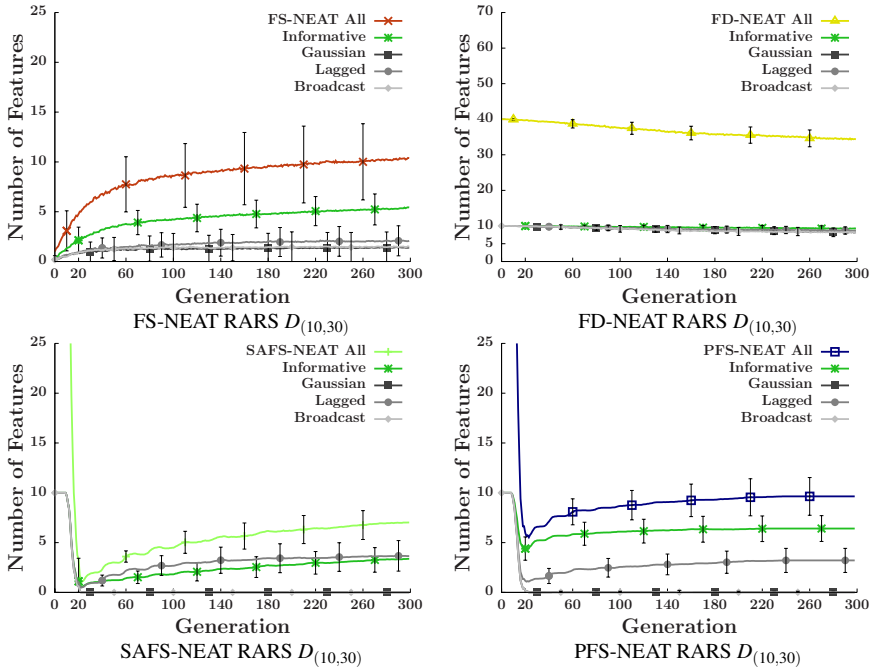


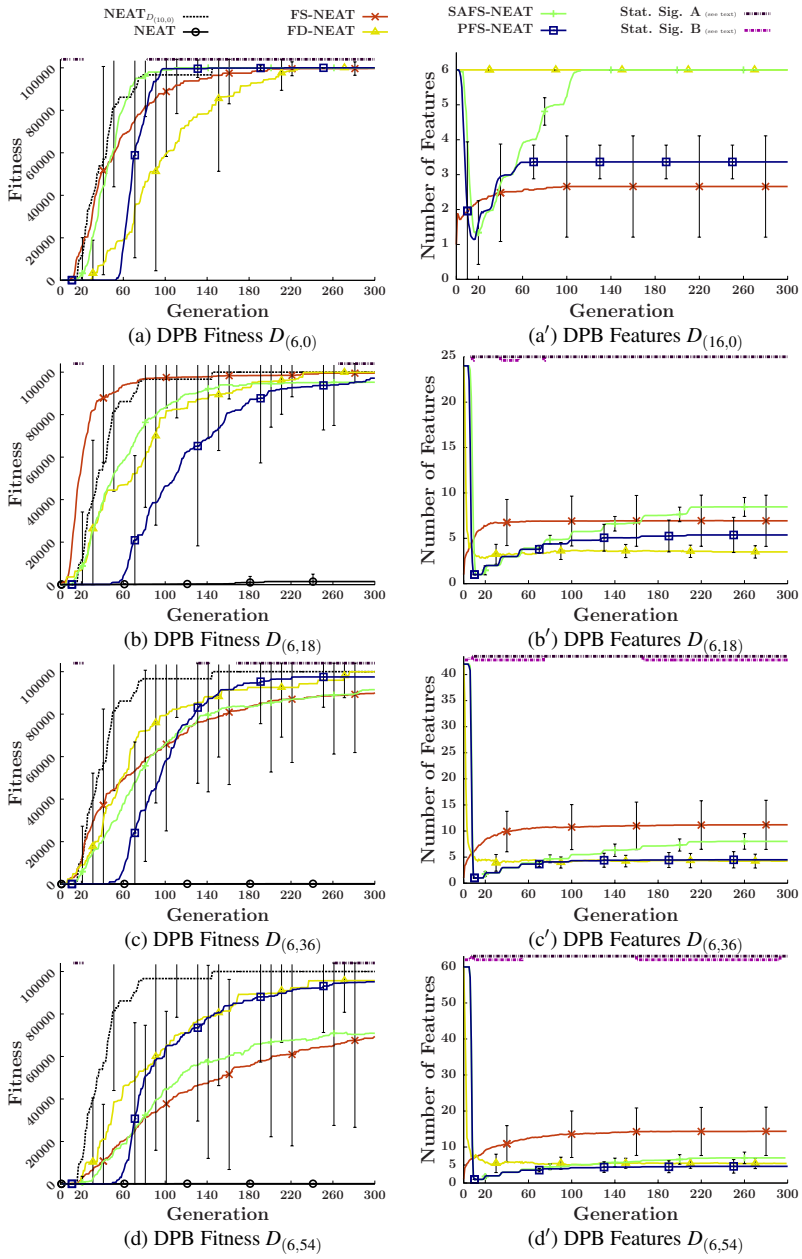
Fig. 7: Selected feature subset compositions for each of the four feature selection algorithms in the RARS $D_{(10,30)}$ scenario. The "All" lines denote the average subset size at each generation, "Informative" denotes the number of relevant features selected, while the other three lines give the counts of each type of additional feature selected. Error bars denote standard deviation.

4.2.2 Double Inverted Pendulum Balancing

The left column of Figure 8 shows the average fitness performance of the five algorithms as the number of additional features increases in the DPB environment. Fitness indicates the number of time steps that both pendulums remained balanced on the cart, with 100,000 being the maximum possible duration in our setup. Similar to the results from the RARS experiments, the performance of NEAT on the scenario with no additional features, $D_{(6,0)}$, is shown as a reference. The statistical significance bars at the top of the fitness plots have the same meaning as described above for the RARS environment, but use ($p < 0.005$) in this environment. We see in Figure 8(b) – (d) that additional features have an even greater negative impact on the NEAT algorithm than they did in the RARS environment, and further motivate the need for effective feature selection algorithms to aid policy search. In this environment PFS-NEAT does not dominate all of the competing feature selection algorithms. In fact, FD-NEAT almost always (the slight exception being a few early generations) produces policies that are at least as fit as PFS-NEAT in all scenarios. PFS-NEAT is able to arrive at policies that are not statistically significantly worse than the reference curve by the end of 300 generations, so it is still an effective feature selection algorithm in this environment. Both FS-NEAT and SAFS-NEAT are better able to cope with the additional sensors in this setting, but their performance still diminishes when 36 or 54 such features are in the environment, as shown in panels (c) and (d). Both FD-NEAT and PFS-NEAT show a slightly negative trend as the number of features increases, but this trend is not as strong as that of

the other algorithms in comparison. The error bars in these plots show that all of the feature selection algorithms have large variances that shrink during the course of evolution, but not to the extent seen in the RARS domain. This is once again due to the dual modality of the results as described in the RARS results. There are some runs where feature selection fails to identify a good feature subset or the policy search does not encounter a successful policy in the 300 generations of the experiments. These bad outcomes happen more or less frequently depending on the algorithm, with PFS-NEAT and FD-NEAT suffering fewer of them as compared to FS-NEAT and SAFS-NEAT. NEAT has small error bars since all runs either performed uniformly well as in Figure 8(a) or uniformly poorly as in (b) – (d). It is worth pointing out the excellent performance of FS-NEAT on $D_{(6,18)}$, as shown on panel (b), where it improves even faster than the baseline NEAT with no additional sensors. This at first may seem counter-intuitive, but after examining the networks produced by FS-NEAT we observed that many NNs connected the inputs to only specific output nodes. NEAT networks always begin fully connected, and thus have a larger number of connections which may be mutated, increasing the policy search space and reducing the chances of evolving a working policy early in the search.

Unlike the RARS environment, PFS-NEAT shows very little improvement in the starting generations in DPB. Due to feature interactions in this domain, several features must first be included in the subset before learning progress can be made. This can be seen by examining the plots in the right column of Figure 8. Only after PFS-NEAT is able to select around 3 features (approximately generation 55), does its policy performance increase. We also note that we limit Best-First search to select one feature at most per selection round in this environment. If not, the first selected subset often contained an irrelevant feature that hampered the converged performance of the algorithm. This limitation could be mitigated if features were allowed to be eliminated from previously selected subsets in PFS-NEAT, and could be the subject of a future extension. Once an appropriate feature set is determined, PFS-NEAT achieves a near optimal fitness. The Statistical Significance A & B curves of plots (a′) – (d′) are measured in the same way as the right column of Figure 6 in the RARS environment. Once again, we see that PFS-NEAT is able to select subsets with a significantly greater fraction of informative features as compared to the other feature selection algorithms, though several of the other algorithms tend to select more lagged sensors than PFS-NEAT. These results show that the Best-First search strategy employed by PFS-NEAT can be used in environments with feature interaction, but it can result in periods of slow policy improvement while samples are gathered to determine an informative subset. These results also help explain why FD-NEAT and FS-NEAT both initially improve faster than PFS-NEAT in all DPB scenarios. Both of these algorithms are able to quickly arrive at small feature subsets containing useful information about how the problem works. We will examine these subset compositions in more detail in Figure 9. FD-NEAT benefits from a very aggressive remove connection weight in this environment, which allows it to rapidly identify a subset of about 5 features on average. A high remove connection weight actually reduced the performance of FD-NEAT in RARS. One explanation for this discrepancy is that RARS requires more specific connection weight schemes given an appropriate set of input nodes, meaning that more generations must be spent evolving given subset size than in the DPB environment.



(a) DPB Fitness $D_{(6,0)}$ (a′) DPB Features $D_{(16,0)}$

(b) DPB Fitness $D_{(6,18)}$ (b′) DPB Features $D_{(6,18)}$

(c) DPB Fitness $D_{(6,36)}$ (c′) DPB Features $D_{(6,36)}$

(d) DPB Fitness $D_{(6,54)}$ (d′) DPB Features $D_{(6,54)}$

Fig. 8: Plots (a) – (d) (left column) denote the average fitness of the algorithms in comparison during the first 300 generations of evolution for each of the DPB scenarios described in Table 1. Plots (a′) – (d′) (right column) show the corresponding selected feature subset sizes of the four feature selection algorithms in comparison. Curves in all plots follow the key at the top of the page. Plots (b) – (d) contain a dotted reference curve replicating the performance of NEAT on $D_{(6,0)}$ (the scenario using only informative features). The Stat. Sig. A and B curves are shown when their conditions are met with $p < 0.005$ in a given generation (see text for details). Error bars denote the standard deviation of the 100 runs at each point.

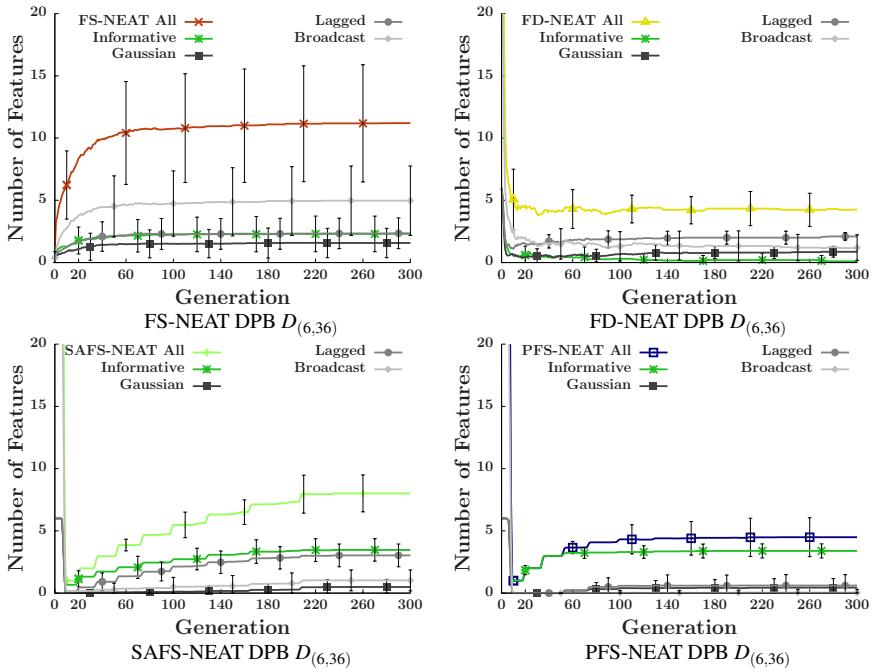


Fig. 9: Selected feature subset compositions for each of the four feature selection algorithms in the DPB $D_{(6,36)}$ scenario. The "All" lines denote the average subset size at each generation, "Informative" denotes the number of relevant features selected, while the other three lines give the counts of each type of additional feature selected. Error bars denote standard deviation.

Figure 9 examines the composition of the feature subsets selected by each of the feature selection algorithms for $D_{(6,36)}$ in the DPB environment. In this scenario, FD-NEAT and PFS-NEAT select similar numbers of features as seen in Figure 8(c′), but these subsets are comprised of overwhelmingly distinct features. We can clearly see that PFS-NEAT selects several informative sensors as soon as it can (given the stagnation parameters), and selects very few other features for the remainder of the evolutionary policy search. The other three algorithms also exhibit a selection plateau around generation 100, where they are able to solve many of the runs after suitably informative subsets have been constructed. The other three algorithms also tend to select lagged sensors, and especially FD-NEAT, which prefers them over the designed informative sensors. From the fitness results of FD-NEAT we can conclude that several of the lagged sensors are useful to the NEAT process and can be more useful to the policy search than the informative sensors. Despite their usefulness, PFS-NEAT does not select them since they do not evaluate highly on the 1-step reward and transition relevance measures used to guide the feature subset search. This result serves to illustrate a general limitation of the PFS-NEAT feature evaluation heuristic: if an environment only has a delayed reward signal or highly stochastic transition dynamics, the feature relevance determination can be inaccurate and lead to poor performance. SAFS-NEAT and FS-NEAT both select lagged sensors while following their selection heuristics, but both prefer broadcast sensors. Selecting both informative and lagged sensors early allows for rapid fitness improvements in the early generations, but the addition of too many of these additional sensors ultimately impedes the progress of the policy search in this environment.

4.2.3 Results Summary

The results presented in Sections 4.2.1 & 4.2.2 have served to illustrate several key points:

(i) additional features can prohibit the learning of a good policy;
(ii) PFS-NEAT is able to effectively exclude many additional features from the search space, leading to good selected feature subsets and fit policies;
(iii) PFS-NEAT is more generally effective than FS-NEAT, SAFS-NEAT, and FD-NEAT, though there are situations where one or more may yield comparable results.

From these points we can conclude that PFS-NEAT can be successfully used to constrain a policy search space by reducing the number of input features to consider. This algorithm has been demonstrated to allow the NEAT genetic policy search technique to scale to environments with high-dimensional state spaces.

5 Discussion and Conclusion

In this work, we have proposed a novel feature selection framework for genetic policy search algorithms, PFS-NEAT, that automatically discovers a good set of features for learning a control policy. To achieve this, it interleaves a Best-First search strategy with the NEAT policy search algorithm, and applies a computationally efficient method to evaluate the goodness of features via previously collected samples. We have provided a detailed study on the characteristics of the NEAT algorithm, with and without feature selection, and have demonstrated that feature selection can allow NEAT to scale to problems that include many additional features. We also show that the feature selection framework proposed here is able to consistently select a small subset primarily comprised of relevant features, enabling NEAT to search a restricted subset of policies.

One immediate direction of future work is motivated from our experimental study. It is clear that FS-NEAT performs well when the search space contains more relevant features than irrelevant ones. It and FD-NEAT can both take advantage of sensors that were not directly thought to contribute to successful policies as in the DPB domain. The feature evaluation strategy of PFS-NEAT is able to score features based on their predicted value to a successful policy. We hypothesize that PFS-NEAT could be used in conjunction with the evolutionary selection algorithms to boost the quality of the candidate pool of input features that they have access to, while allowing them to explore many different combinations of these input features. More broadly, we can consider how to incorporate the predictive feature selection idea with other direct policy search algorithms or RL methods in general, outside of the genetic policy search context. There are two main areas that we rely on NEAT for the approach presented in this paper: determining when to select features, and how to insert them into an existing policy function. The stagnation criterion we used must track fitness changes of the champion NN from NEAT's population, something that would not exist in non-genetic policy search algorithms. Instead, for gradient policy search methods such as the one presented by Deisenroth and Rasmussen (2011), we could decide to search for more features once the gradient changes drop below some threshold. Preserving existing performance after incorporating new features depends on the representation of the policy function used by the algorithm, and so new transfer methods may need to be explored. In general, one could create a training data set using the current best policy function, and train the new function which includes the updated feature subset on that data set. There would

be an additional time and computation penalty here, but the samples could be generated synthetically to avoid an additional sampling overhead.

We briefly discussed some alternatives to the stagnation criterion for determining when to select features in Section 3.4. Stagnation is intrinsically tied to fitness in our work, but recent research has shown the validity of searching for non-fitness criteria, and these methods may present a viable means to further the subset search (Mouret and Doncieux, 2012). Our fundamental goal with this work is to enable genetic policy search algorithms to find fit policies by constraining the policy search to consider only subsets of the feature space that are found to be relevant. If a diversity procedure like Novelty Search was adopted (Lehman and Stanley, 2011), it may be possible to consider novel behavior-producing networks induced by different feature subsets. This may allow genetic policy search to rapidly find fit policies by implicitly performing feature selection to promote genetic diversity instead of fitness, and is an area to consider for future study.

The Predictive Feature Selection framework's core iterative process of altering the feature space can be viewed as automatically guiding the evolutionary process through a series of simpler approximations to arrive at a fit policy for the task of interest. In this way, it bears a resemblance to the deception avoidance work of Gomez and Miikkulainen (1997), and is related to the larger world of transfer learning in RL (Taylor and Stone, 2009). PFS-NEAT is optimistic that the genetic search will benefit from the inclusion of features found to be relevant by the evaluation criteria, but this process does not actively seek to avoid deceptive learning situations as done in Gomez and Miikkulainen (1997). An open future direction is to determine a best progression of feature subsets, and may be able to leverage concepts and techniques from the emerging field of Curriculum Learning (Bengio et al, 2009).

As illustrated in this work, feature selection can lead to significant benefits to genetic policy search algorithms, allowing to scale them to high-dimensional state spaces such as those found in many real world control problems. There are many possible areas of inclusion besides the direct policy search case presented in this work, and we believe investigating the interaction between RL and feature selection in these different scenarios will lead to breakthroughs in the applicability of RL algorithms.

6 Acknowledgements

This work was performed under 13-RI-CRADA-13, and was supported in part through computational resources provided by the U.S. DoD HPCMP AFRL/RI Affiliated Resource Center. The authors would like to thank the anonymous reviewers for their helpful comments and suggestions, and Kevin Acunto for his work porting the RARS environment to Java.

References

Argall BD, Chernova S, Veloso M, Browning B (2009) A survey of robot learning from demonstration. Robot Auton Syst 57:469–483

Bellman R (2003) Dynamic Programming. Dover Publications

Bengio Y, Louradour J, Collobert R, Weston J (2009) Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning, ACM, pp 41–48

Böhm N, Kkai G, Mandl S (2004) Evolving a heuristic function for the game of tetris. In: Lernen, Wissensentdeckung und Adaptivität (LWA), Humbold-Universität Berlin, pp 118–122

Boutilier C, Dean T, Hanks S (1999) Decision-theoretic planning: Structural assumptions and computational leverage. JAIR 11:1–94

Cannady J (2000) Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In: Proceedings of the 23rd National Information Systems Secuity Conference, pp 1–12

Castelletti A, Galelli S, Restelli M, Soncini-Sessa R (2011) Tree-based variable selection for dimensionality reduction of large-scale control systems. In: Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on, IEEE, pp 62–69

Cliff D, Miller G (1995) Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In: Morn F, Moreno A, Merelo J, Chacn P (eds) Advances in Artificial Life, Lecture Notes in Computer Science, vol 929, Springer Berlin Heidelberg, pp 200–218, DOI 10.1007/3-540-59496-5_300

Deisenroth M, Rasmussen C (2011) Pilco: A model-based and data-efficient approach to policy search. In: Getoor L, Scheffer T (eds) Proceedings of the 28th International Conference on Machine Learning (ICML-11), ACM, New York, NY, USA, ICML '11, pp 465–472

Devijver P, Kittler J (1982) Pattern Recognition: A Statistical Approach. Prentice Hall International

Dietterich TG (1998) The maxq method for hierarchical reinforcement learning. In: In Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann, pp 118–126

Diuk C, Li L, Leffler B (2009) The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In: Bottou L, Littman M (eds) Proceedings of the 26th International Conference on Machine Learning, Omnipress, Montreal, pp 249–256

Doroodgar B, Nejat G (2010) A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments. In: 2010 IEEE Conference on Automation Science and Engineering (CASE), pp 948–953

Ernst D, Geurts P, Wehenkel L (2005) Tree-based batch mode reinforcement learning. JMLR 6:503–556

Goldberg DE, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp 41–49

Gomez F, Miikkulainen R (1997) Incremental evolution of complex general behavior. Adaptive Behavior 5:5–317

Gomez FJ, Miikkulainen R (1999) Solving non-markovian control tasks with neuroevolution. In: In Proceedings of the 16th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp 1356–1361

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. Journal of Machine Learning Research 3:1157–1182

Guyon I, Weston J, Barnhill S, Vapnik V (2002) Gene selection for cancer classification using support vector machines. Machine Learning 46:389–422

Hachiya H, Sugiyama M (2010) Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information. In: Proceedings of the ECML, pp 474–489

Hall M (1999) Correlation based feature selection for machine learning. PhD thesis, University of Waikato, Dept. of Computer Science

Jolliffe IT (2010) Principal Component Analysis, Second Edition. Springer

Jung T, Stone P (2009) Feature selection for value function approximation using bayesian model selection. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, pp 660–675

Knowles JD, Watson RA, Corne DW (2001) Reducing local optima in single-objective problems by multi-objectivization. In: Zitzler E, Thiele L, Deb K, Coello Coello C, Corne D (eds) Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, vol 1993, Springer Berlin Heidelberg, pp 269–283, DOI 10.1007/3-540-44719-9_19

Kolter JZ, Ng AY (2009) Regularization and feature selection in least-squares temporal difference learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp 521–528

Konidaris G, Barto A (2009) Efficient skill learning using abstraction selection. In: Proceedings of the 21st international jont conference on Artifical intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1107–1112

Konidaris G, Kuindersma S, Barto A, Grupen R (2010) Constructing skill trees for reinforcement learning agents from demonstration trajectories. In: NIPS 23, pp 1162–1170

Kveton B, Hauskrecht M, Guestrin C (2006) Solving factored MDPs with hybrid state and action variables. J Artif Int Res 27:153–201

Lazaric A, Restelli M, Bonarini A (2007) Reinforcement learning in continuous action spaces through sequential monte carlo methods. In: Advances in Neural Information Processing Systems, pp 833–840

Lehman J, Stanley KO (2011) Abandoning objectives: Evolution through the search for novelty alone. Evolutionary computation 19(2):189–223

Li L, Walsh TJ, Littman ML (2006) Towards a unified theory of state abstraction for MDPs. In: In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics, pp 531–539

Liu H, Yu L (2005) Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on Knowledge and Data Engineering 17(4):491–502

Loscalzo S, Wright R, Acunto K, Yu L (2012) Sample aware embedded feature selection for reinforcement learning. In: Proceedings of GECCO, pp 879–886

Mahadevan S (2005) Representation policy iteration. In: Proceedings of the Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05), AUAI Press, Arlington, Virginia, pp 372–379

March JG (1991) Exploration and exploitation in organizational learning. In: Organizational Science, vol 2(1), pp 71–87

Melo FS, Lopes M (2008) Fitted natural actor-critic: A new algorithm for continuous state-action MDPs. In: ECML/PKDD(2), pp 66–81

Mouret JB, Doncieux S (2012) Encouraging behavioral diversity in evolutionary robotics: An empirical study. Evol Comput 20(1):91–133, DOI 10.1162/EVCO_a_00048

Nouri A, Littman M (2010) Dimension reduction and its application to model-based exploration in continuous spaces. Machine Learning 81:85–98

Parr R, Painter-Wakefield C, Li L, Littman ML (2007) Analyzing feature generation for value-function approximation. In: ICML, pp 737–744

Pazis J, Lagoudakis MG (2009) Binary action search for learning continuous-action control policies. In: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, New York, NY, USA, ICML '09, pp 793–800

Petrik M, Taylor G, Parr R, Zilberstein S (2010) Feature selection using regularization in approximate linear programs for markov decision processes. In: Proceedings of the 27th International Conference on Machine Learning, pp 871–878

Powell WB (2011) Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition. Wiley

Puterman ML (1994) Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience

Servin A, Kudenko D (2008) Multi-agent reinforcement learning for intrusion detection: A case study and evaluation. In: Proceedings of the European Conference on Artificial Intelligence, pp 873–874

Sher GI (2012) Handbook of Neuroevolution Through Erlang. Springer

Stanley KO, Miikkulainen R (2002) Efficient reinforcement learning through evolving neural network topologies. In: Proceedings of GECCO, pp 569–577

Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press

Tan M, Hartley M, Bister M, Deklerck R (2009) Automated feature selection in neuroevolution. Evolutionary Intelligence 1(4):271–292

Tan M, Deklerck R, Jansen B, Cornelis J (2012) Analysis of a feature-deselective neuroevolution classifier (fd-neat) in a computer-aided lung nodule detection system for ct images. In: Soule T, Moore JH (eds) GECCO (Companion), ACM, pp 539–546

Taylor ME, Stone P (2009) Transfer learning for reinforcement learning domains: A survey. JMLR 10:1633–1685

Tesauro G, Das R, Chan H, Kephart JO, Levine D, III FLR, Lefurgy C (2007) Managing power consumption and performance of computing systems using reinforcement learning. In: NIPS

Vigorito CM, Barto AG (2009) Incremental structure learning in factored MDPs with continuous states and actions. Tech. rep., University of Massachusetts Amherst - Department of Computer Science

Watkins CJCH, Dayan P (1992) Technical note q-learning. Machine Learning 8:279–292

Whiteson S, Stone P (2006) Evolutionary function approximation for reinforcement learning. Journal of Machine Learning Research 7:877–917

Whiteson S, Stone P, Stanley KO (2005) Automatic feature selection in neuroevolution. In: Proceedings of GECCO, pp 1225–1232

Wright R, Loscalzo S, Yu L (2011) Embedded incremental feature selection for reinforcement learning. In: ICAART 2011 - Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Volume 1 - Artificial Intelligence, Rome, Italy, January 28-30, 2011, pp 263–268

Xu L, Yan P, Chang T (1988) Best first strategy for feature selection. In: Proceedings of the Ninth International Conference on Pattern Recognition, pp 706–708